Probabilistic Logic Programming

IRSS 2014

Angelika Kimmig
angelika.kimmig@cs.kuleuven.be

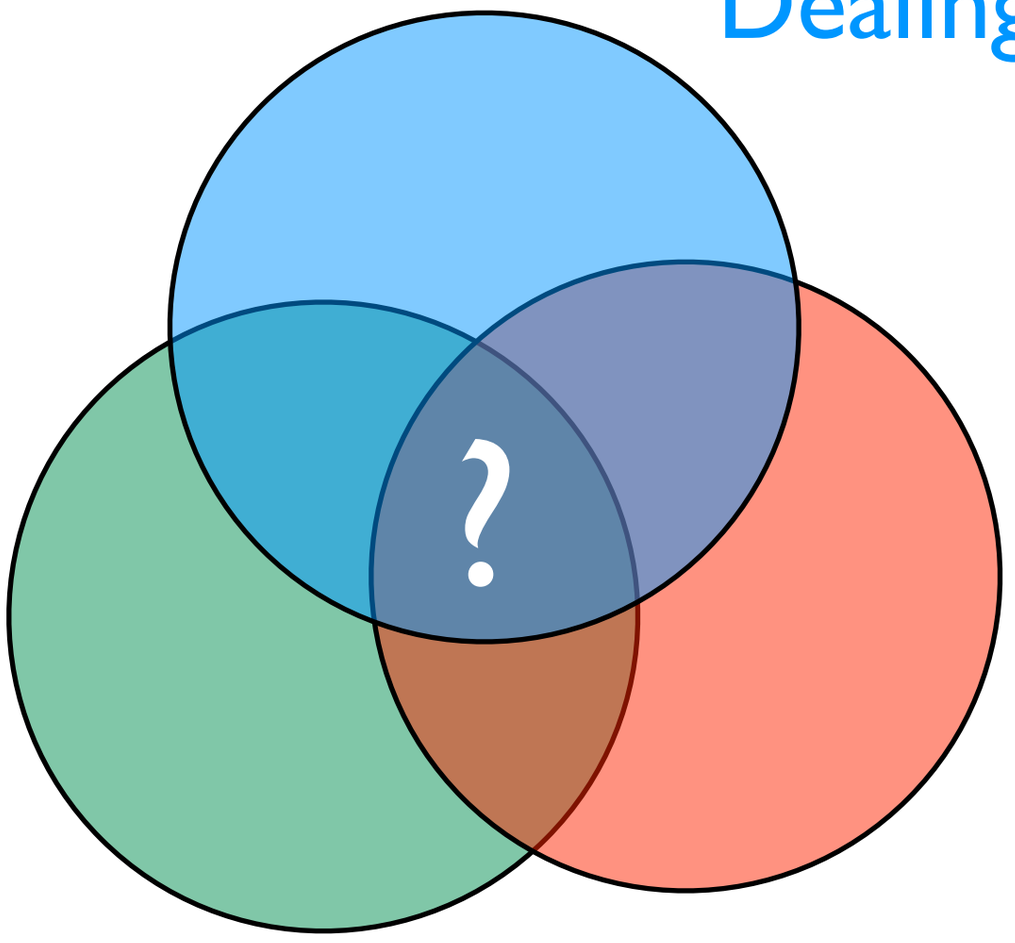
Probabilistic Logic Programming

Thanks to Luc De Raedt & many others working on PLP and especially ProbLog!

A key question in AI:

Dealing with uncertainty

Reasoning with relational data



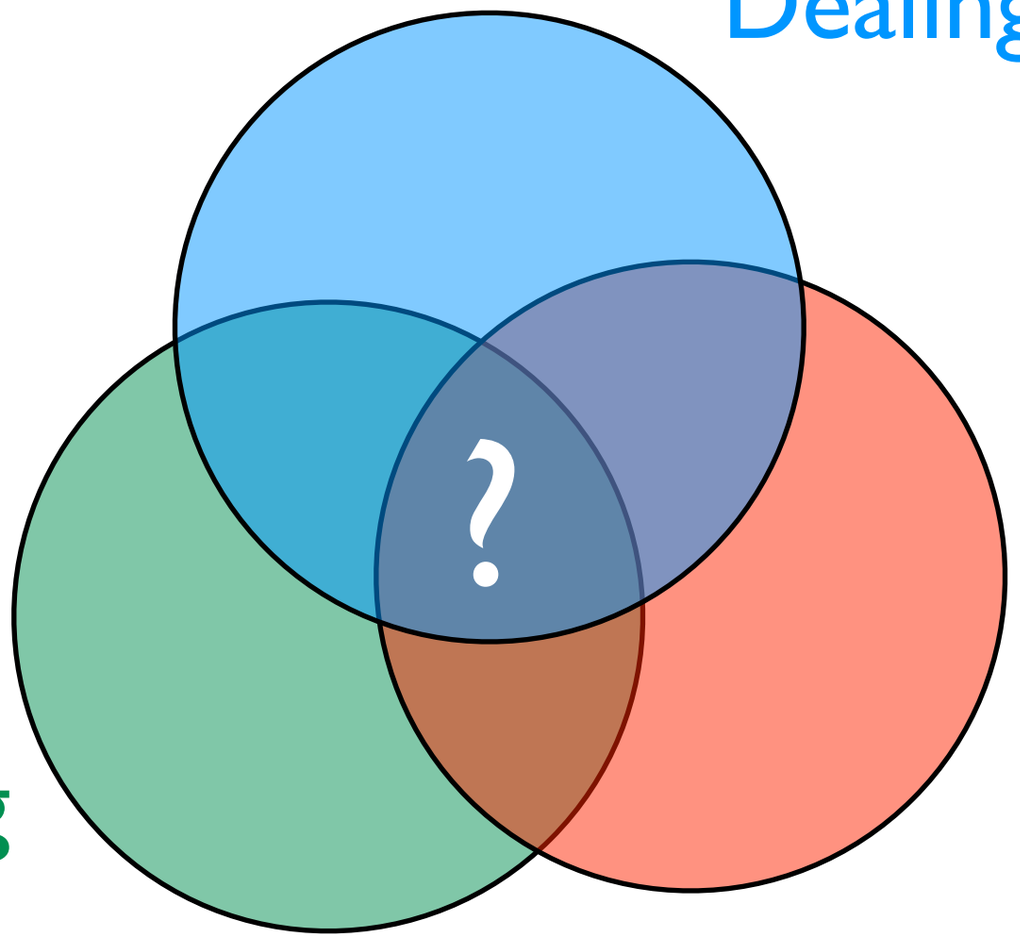
Learning

A key question in AI:

Dealing with uncertainty

Reasoning with relational data

- logic
- databases
- programming
- ...



Learning

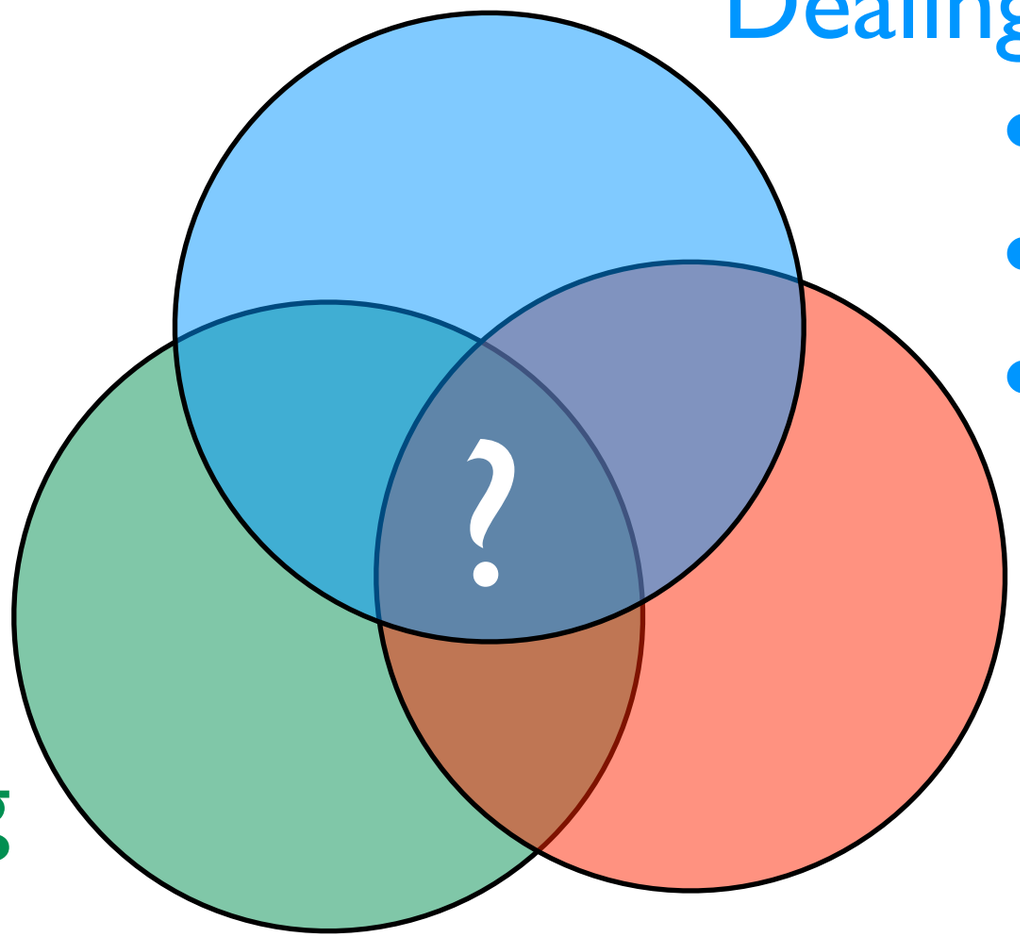
A key question in AI:

Dealing with uncertainty

- probability theory
- graphical models
- ...

Reasoning with relational data

- logic
- databases
- programming
- ...



Learning

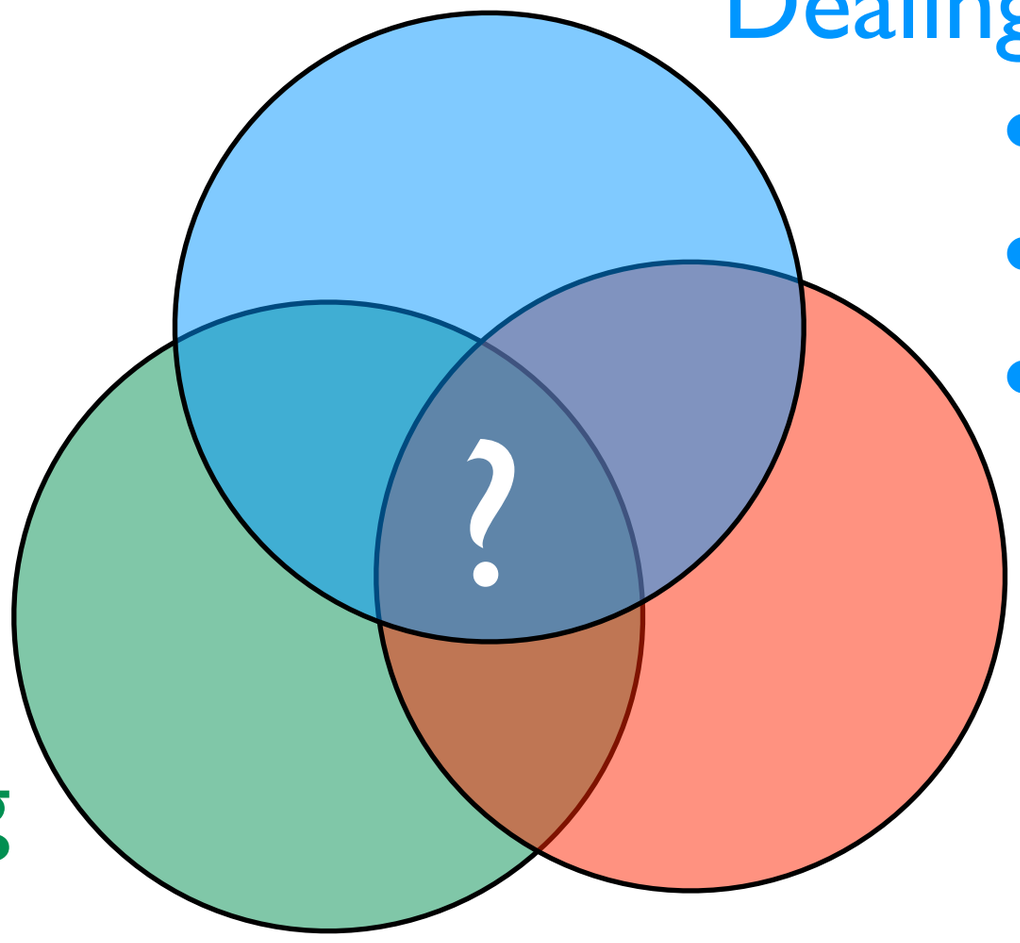
A key question in AI:

Dealing with uncertainty

- probability theory
- graphical models
- ...

Reasoning with relational data

- logic
- databases
- programming
- ...



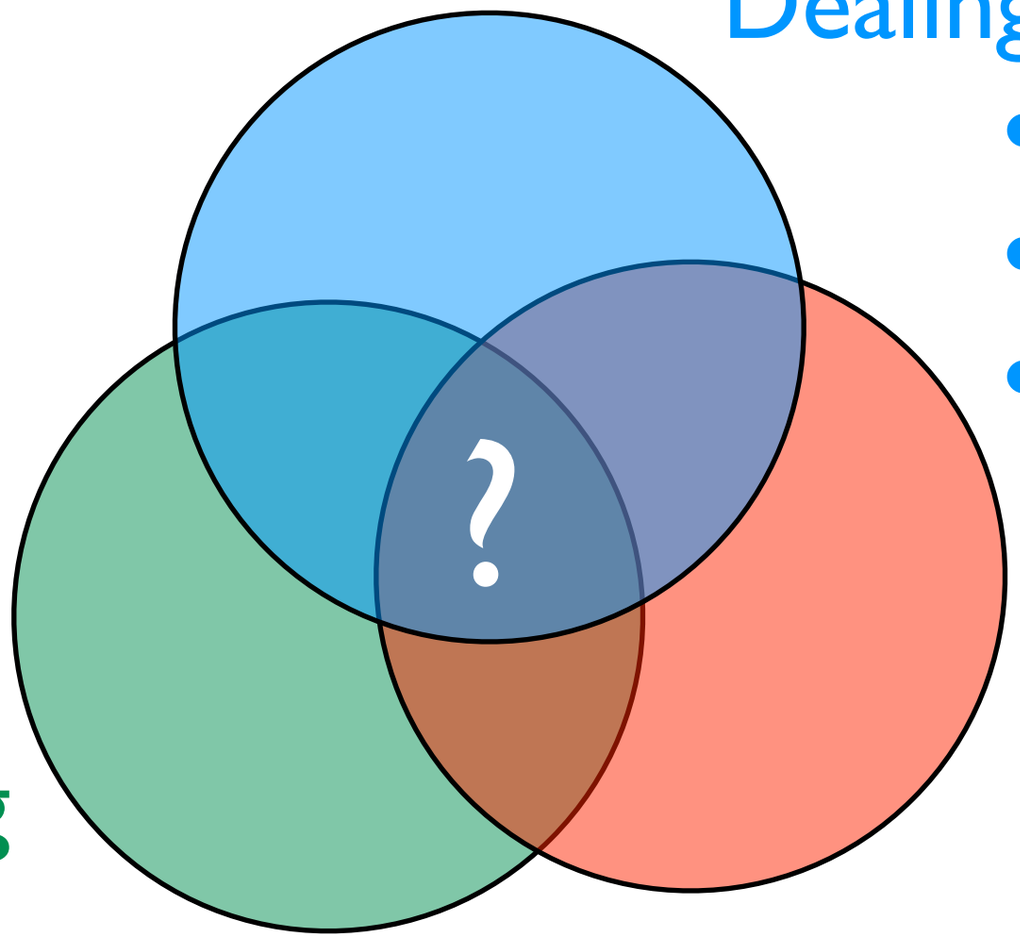
Learning

- parameters
- structure

A key question in AI:

Reasoning with relational data

- logic
- databases
- programming
- ...



Dealing with uncertainty

- probability theory
- graphical models
- ...

Learning

- parameters
- structure

Statistical relational learning, probabilistic logic learning, probabilistic programming, ...

Example: Information Extraction

Recently-Learned Facts twitter Refresh

instance	iteration	date learned	confidence
kelly_andrews is a female	826	29-mar-2014	98.7
investment_next_year is an economic sector	829	10-apr-2014	95.3
shibenik is a geopolitical entity that is an organization	829	10-apr-2014	97.2
quality_web_design_work is a character trait	826	29-mar-2014	91.0
mercedes_benz_cls_by_carlsson is an automobile manufacturer	829	10-apr-2014	95.2
social_work is an academic program at the university rutgers_university	827	02-apr-2014	93.8
dante wrote the book the_divine_comedy	826	29-mar-2014	93.8
willie_aames was born in the city los_angeles	831	16-apr-2014	100.0
kitt_peak is a mountain in the state or province arizona	831	16-apr-2014	96.9
greenwich is a park in the city london	831	16-apr-2014	100.0

NELL: http://rtw.ml.cmu.edu/rtw/

Example: Information Extraction

Recently-Learned Facts twitter Refresh

instance	iteration	date learned	confidence
kelly_andrews is a female	826	29-mar-2014	98.7
investment_next_year is an economic sector	829	10-apr-2014	95.3
shibenik is a geopolitical entity that is an organization	829	10-apr-2014	97.2
quality_web_design_work is a character trait	826	29-mar-2014	91.0
mercedes_benz_cls_by_carlsson is an automobile manufacturer	829	10-apr-2014	95.2
social_work is an academic program at the university rutgers_university	827	02-apr-2014	93.8
dante wrote the book the_divine_comedy	826	29-mar-2014	93.8
willie_aames was born in the city los_angeles	831	16-apr-2014	100.0
kitt_peak is a mountain in the state or province arizona	831	16-apr-2014	96.9
greenwich is a park in the city london	831	16-apr-2014	100.0

instances for many different relations

NELL: http://rtw.ml.cmu.edu/rtw/

Example: Information Extraction

Recently-Learned Facts twitter Refresh

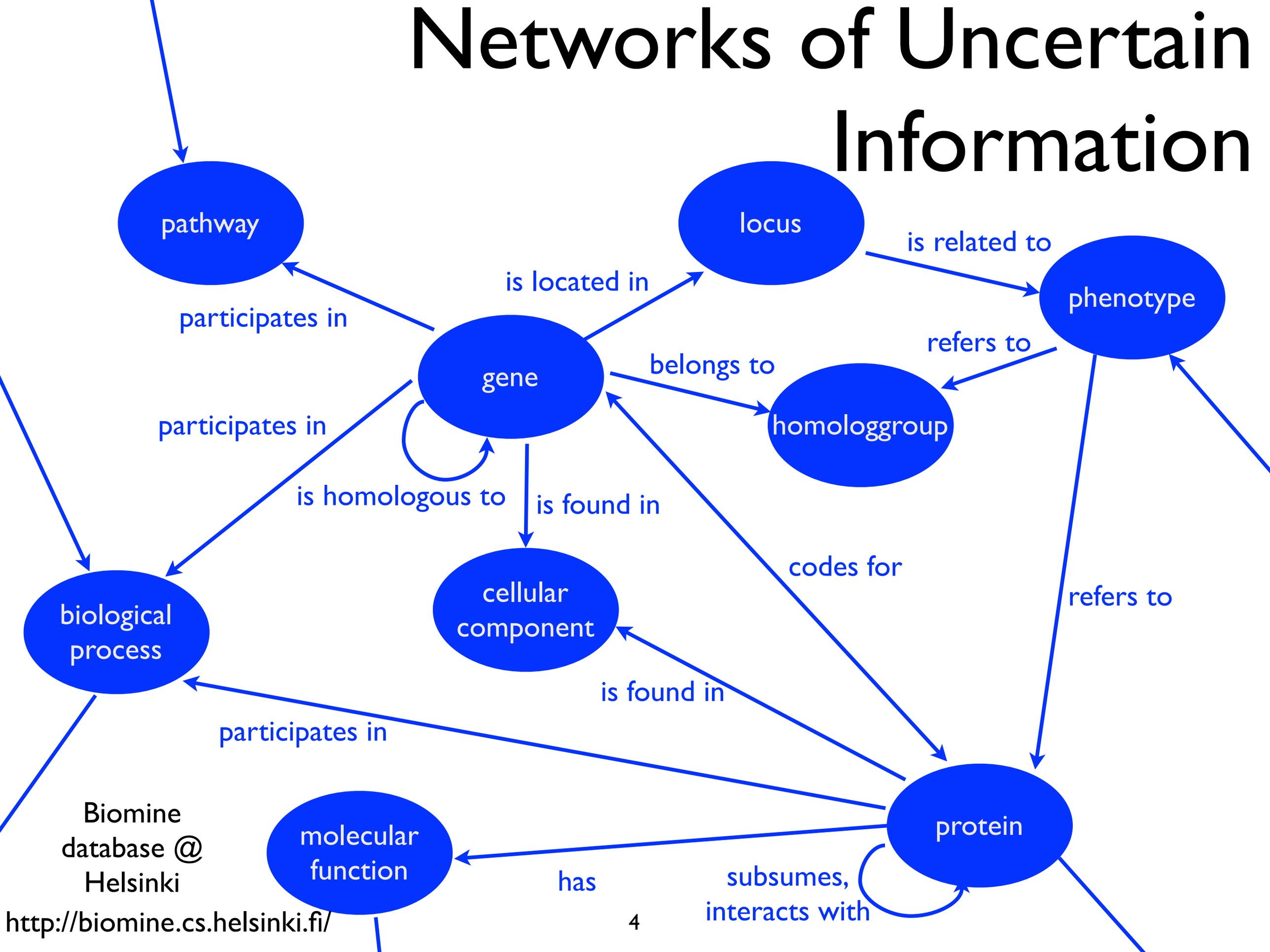
instance	iteration	date learned	confidence
kelly_andrews is a female	826	29-mar-2014	98.7
investment_next_year is an economic sector	829	10-apr-2014	95.3
shibenik is a geopolitical entity that is an organization	829	10-apr-2014	97.2
quality_web_design_work is a character trait	826	29-mar-2014	91.0
mercedes_benz_cls_by_carlsson is an automobile manufacturer	829	10-apr-2014	95.2
social_work is an academic program at the university rutgers_university	827	02-apr-2014	93.8
dante wrote the book the_divine_comedy	826	29-mar-2014	93.8
willie_aames was born in the city los_angeles	831	16-apr-2014	100.0
kitt_peak is a mountain in the state or province arizona	831	16-apr-2014	96.9
greenwich is a park in the city london	831	16-apr-2014	100.0

instances for many different relations

degree of certainty

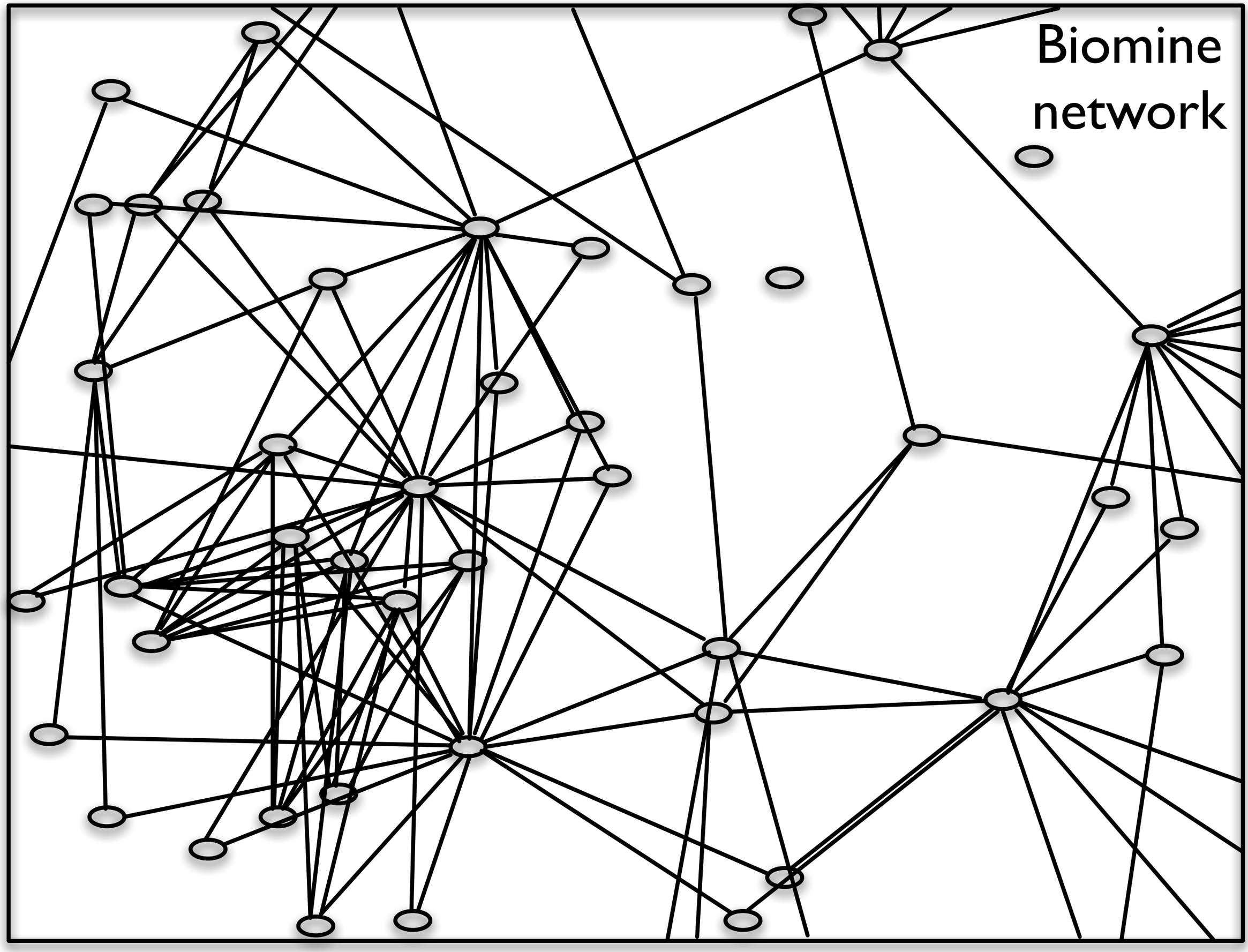
NELL: http://rtw.ml.cmu.edu/rtw/

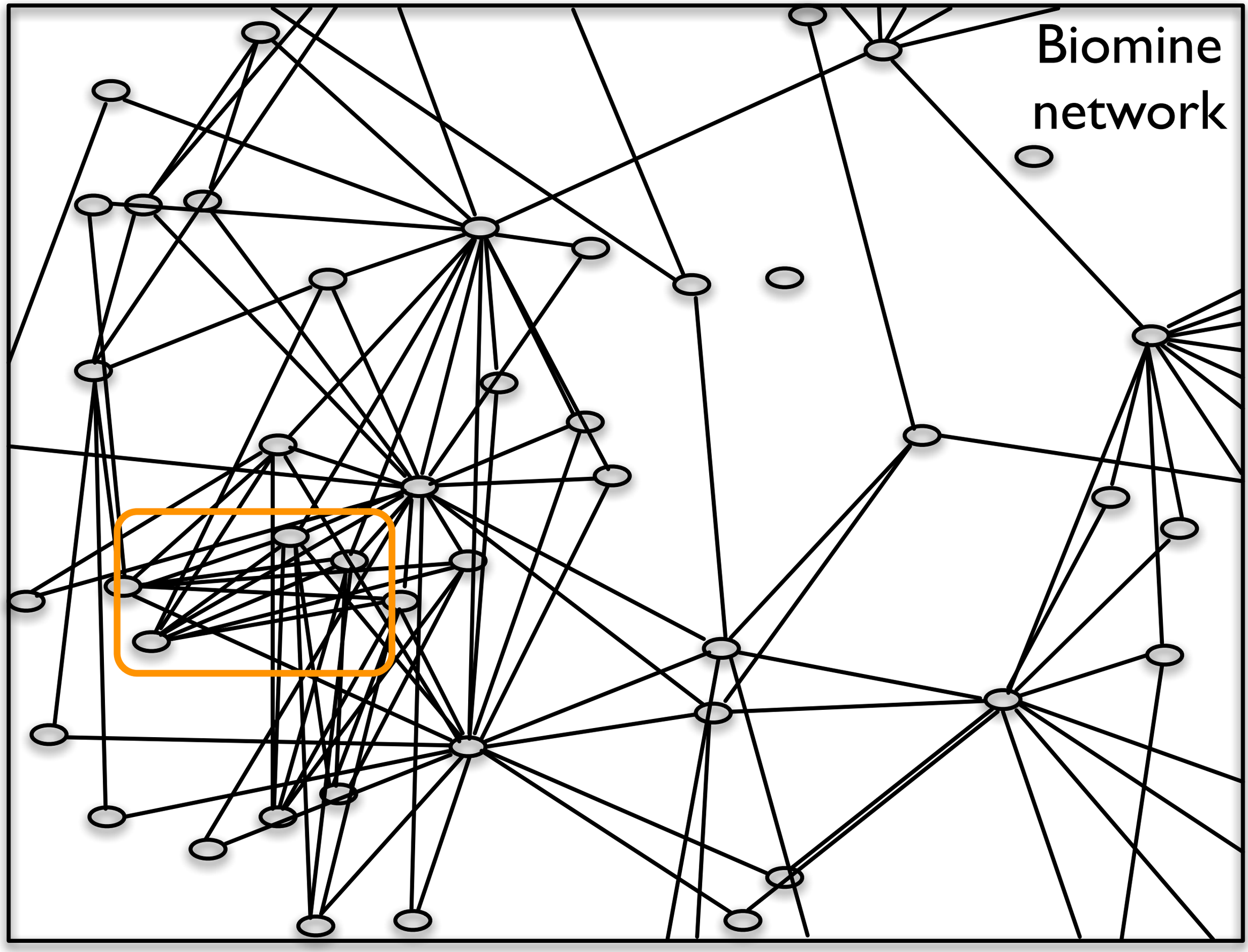
Networks of Uncertain Information



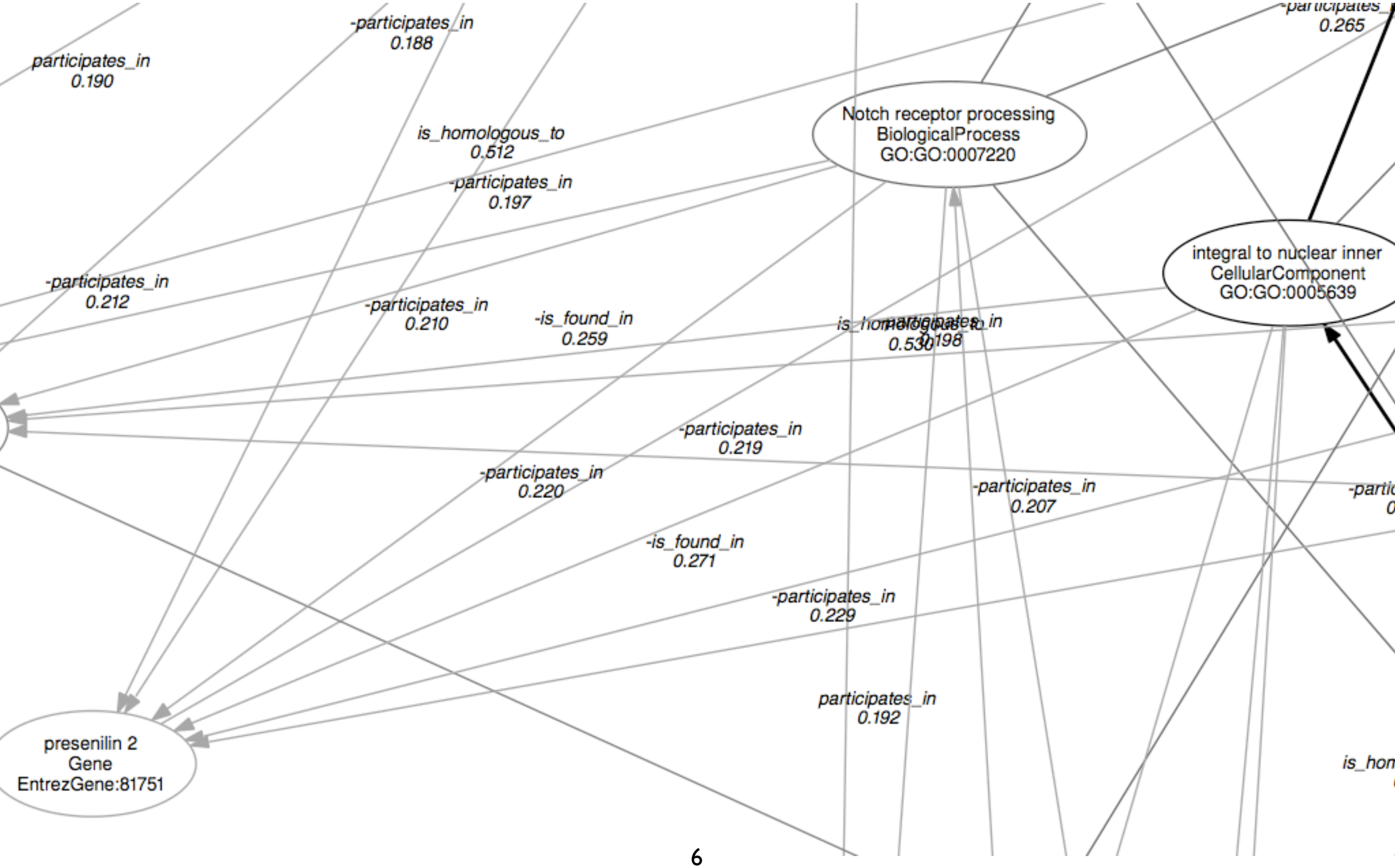
Biomine database @ Helsinki

http://biomine.cs.helsinki.fi/


Biomine
network


Biomine
network

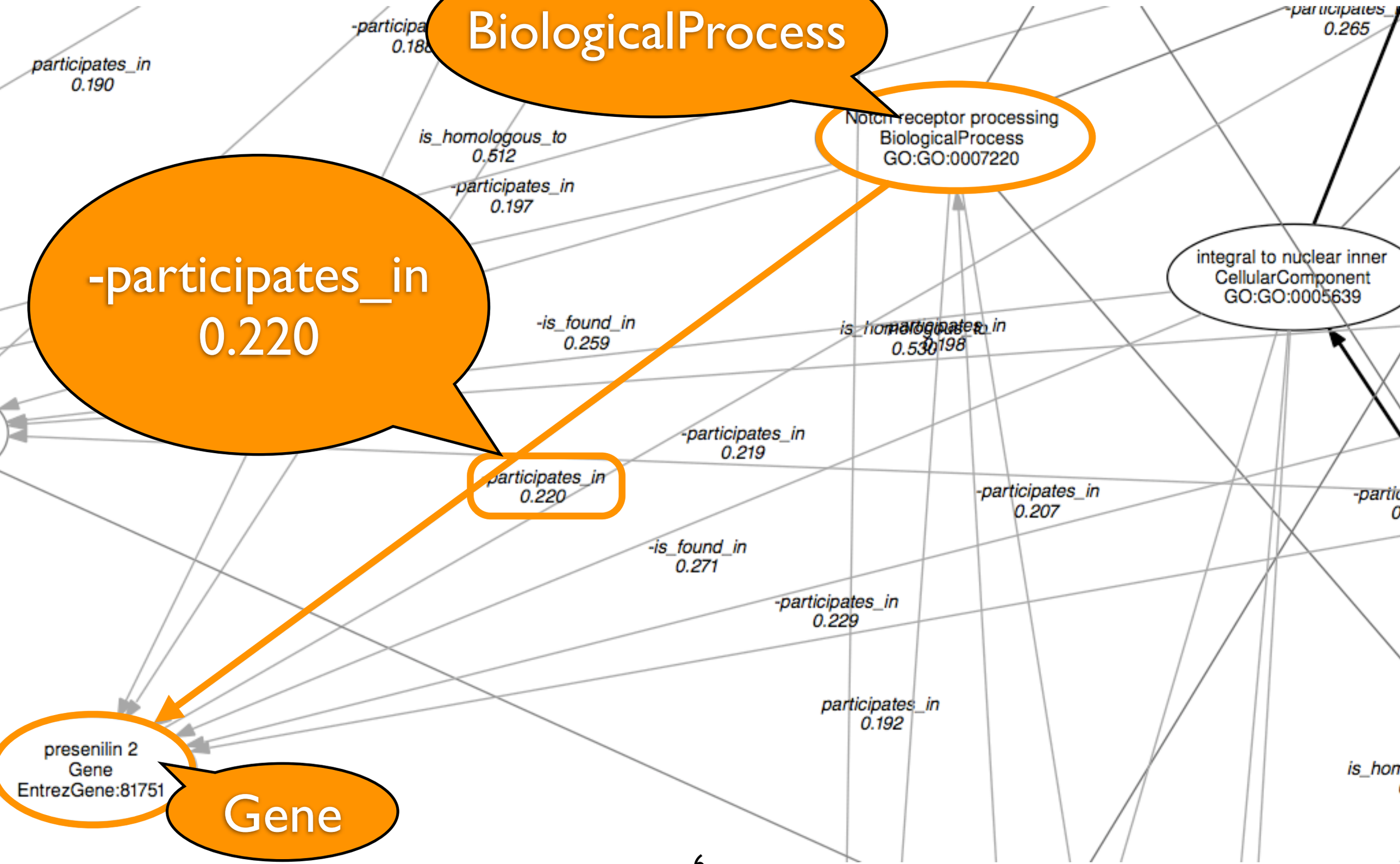
Biomine Network



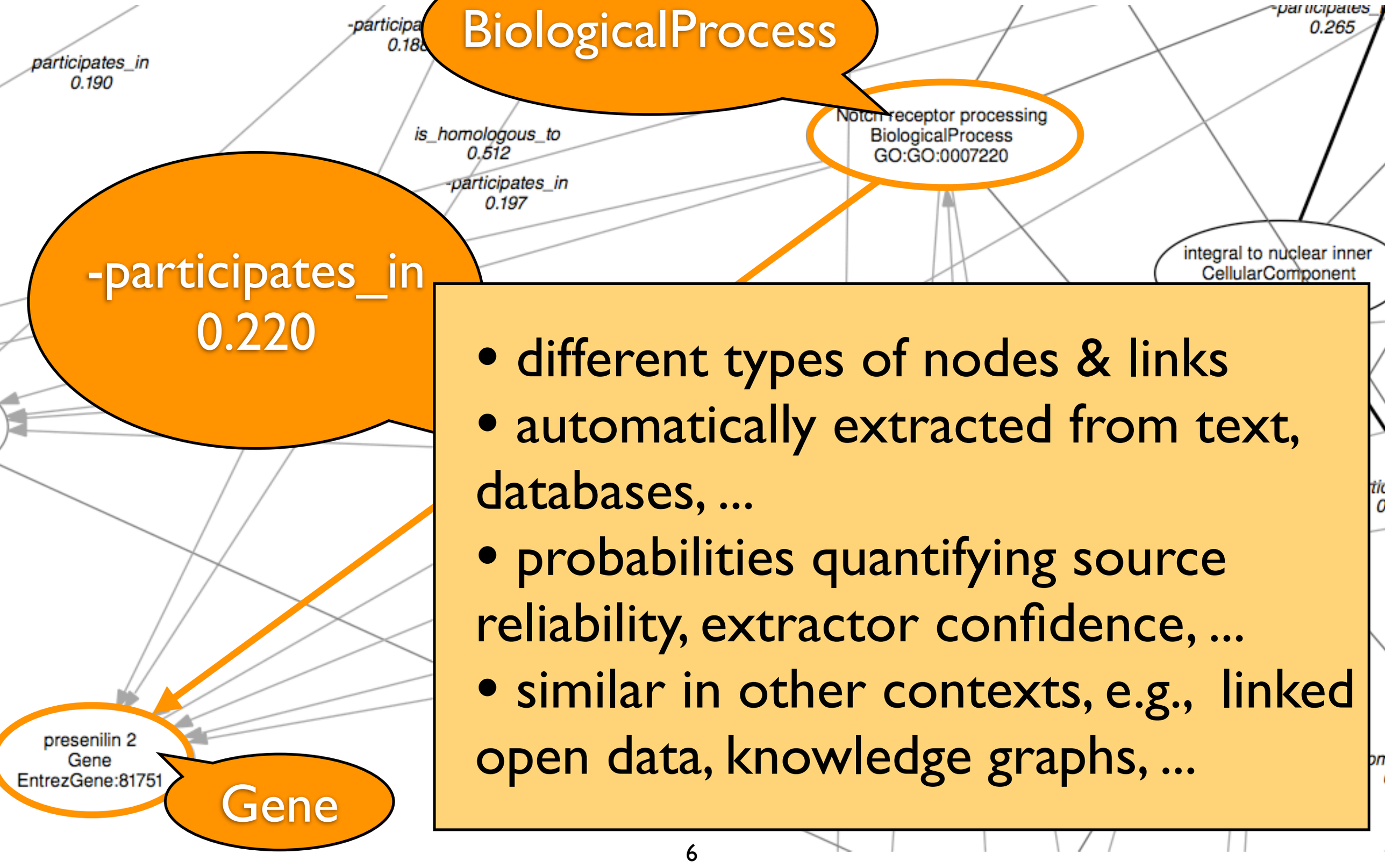
Notch receptor processing
BiologicalProcess
GO:GO:0007220

presenilin 2
Gene
EntrezGene:81751

Biomine Network



Biomine Network

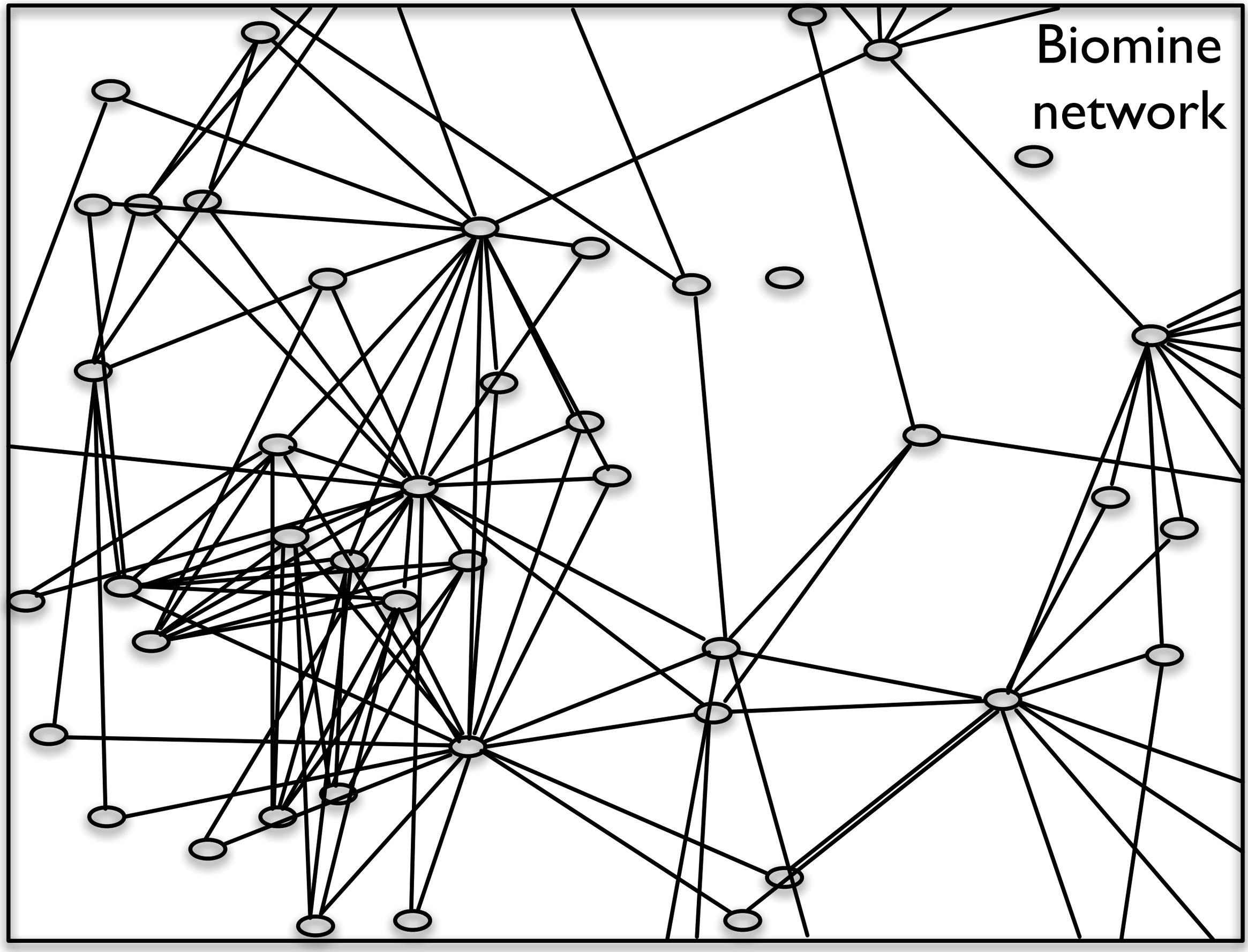


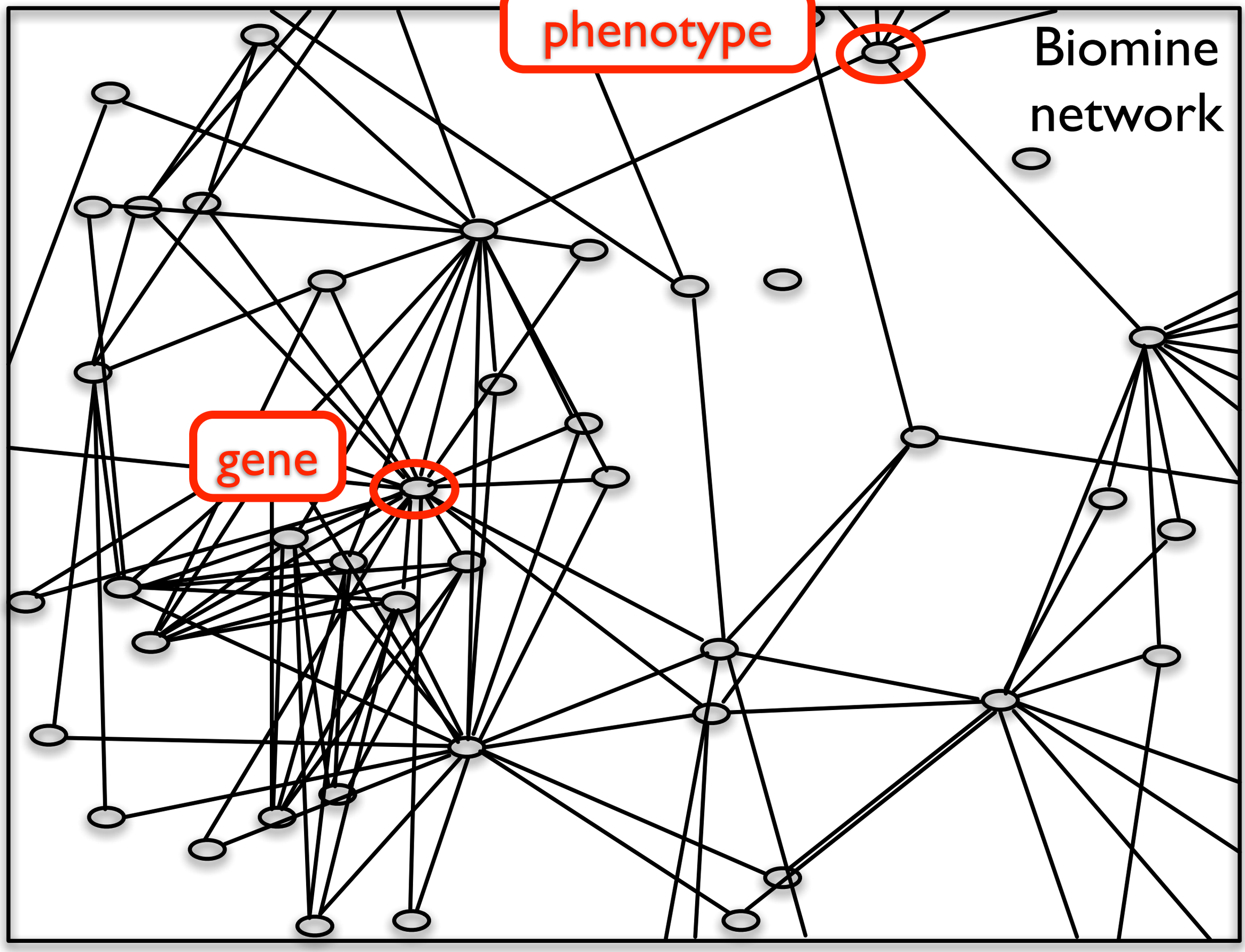
BiologicalProcess

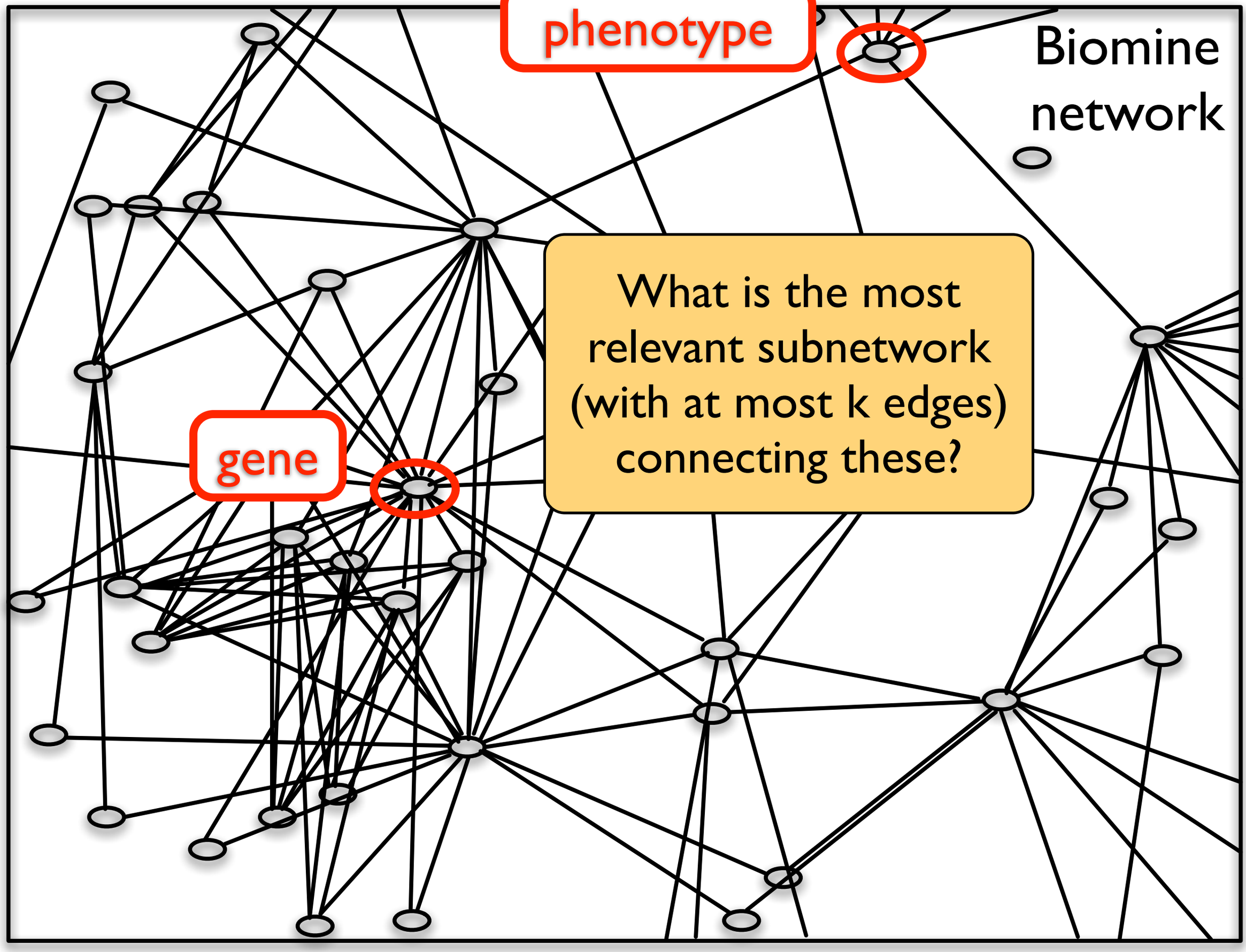
-participates_in
0.220

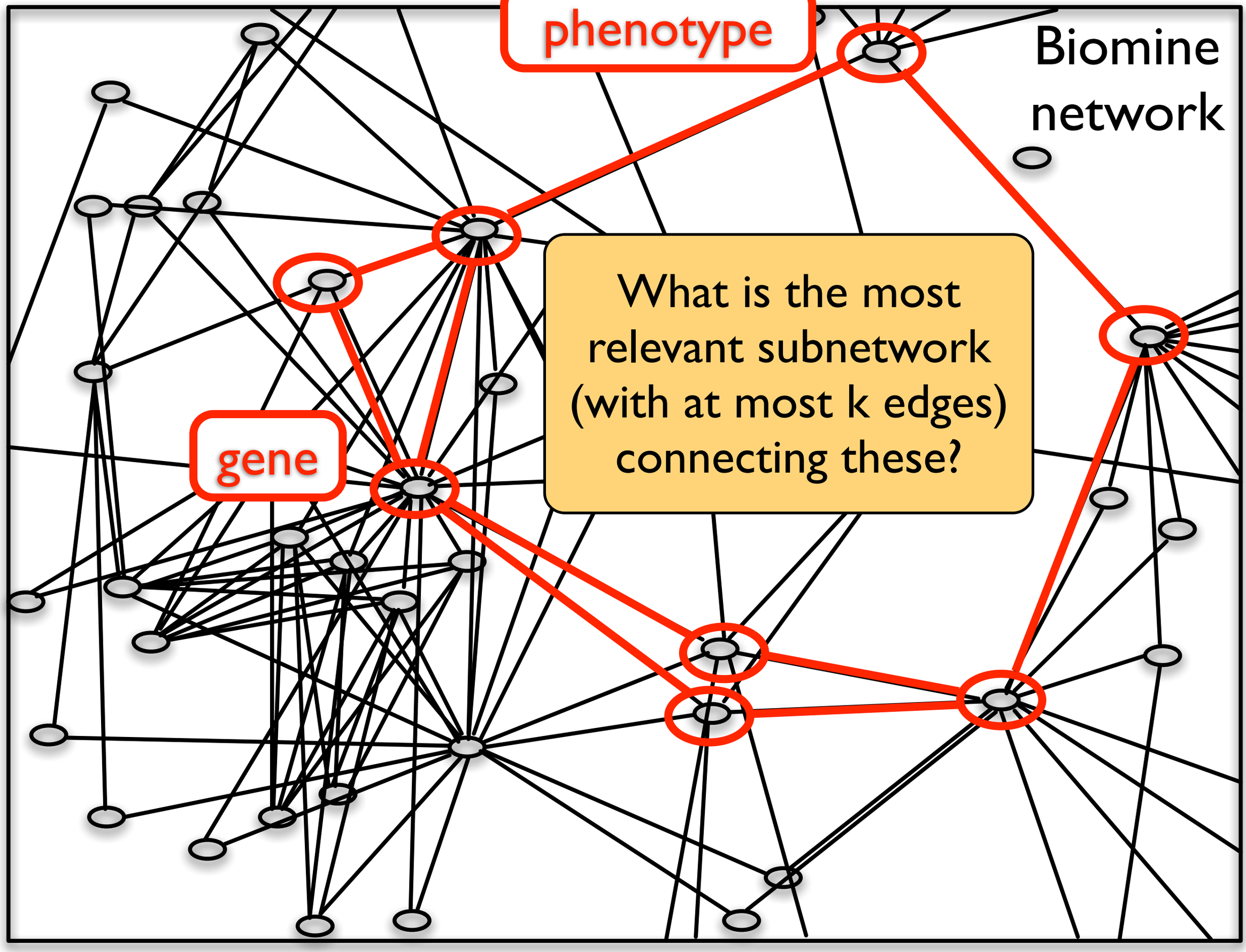
Gene

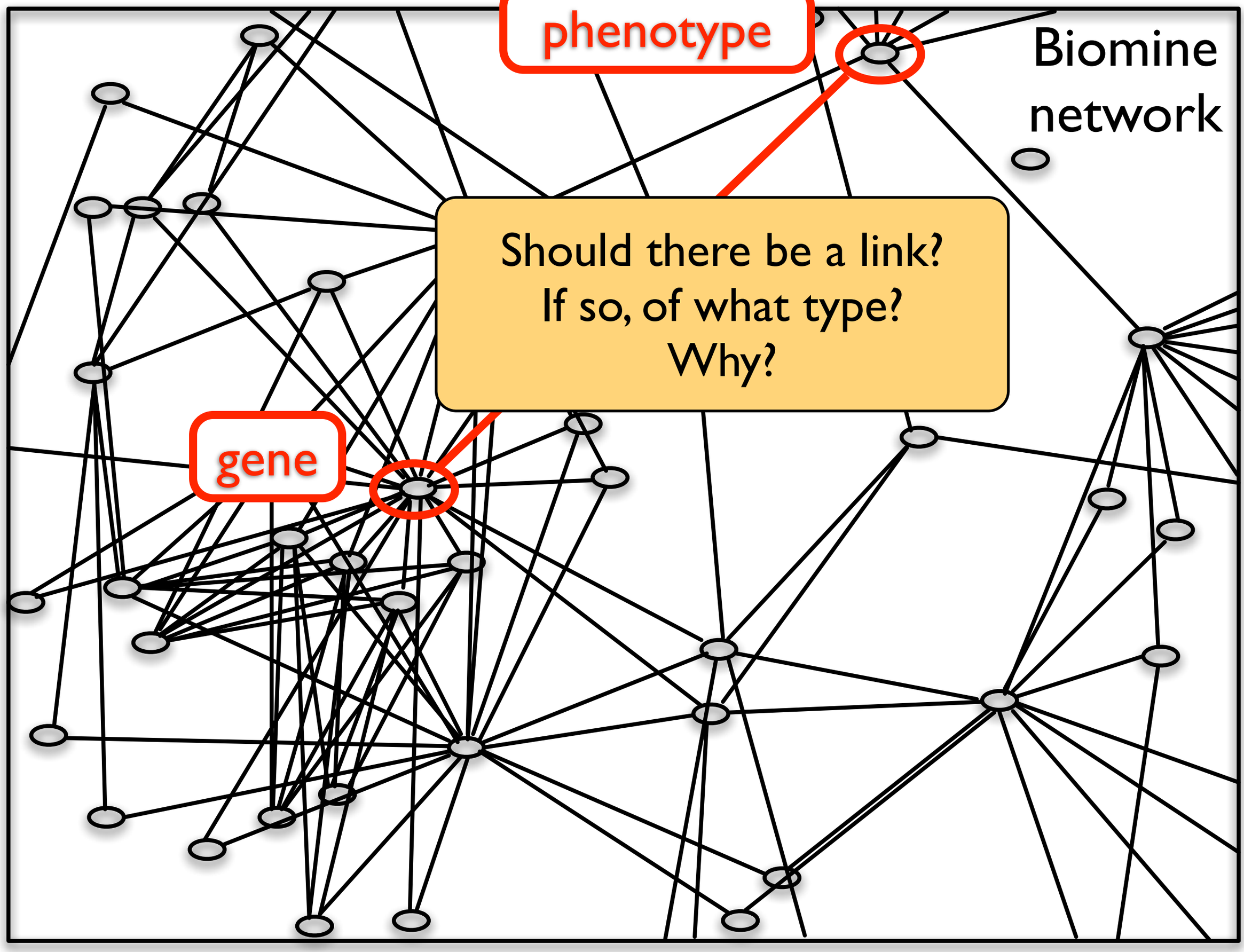
- different types of nodes & links
- automatically extracted from text, databases, ...
- probabilities quantifying source reliability, extractor confidence, ...
- similar in other contexts, e.g., linked open data, knowledge graphs, ...


Biomine
network

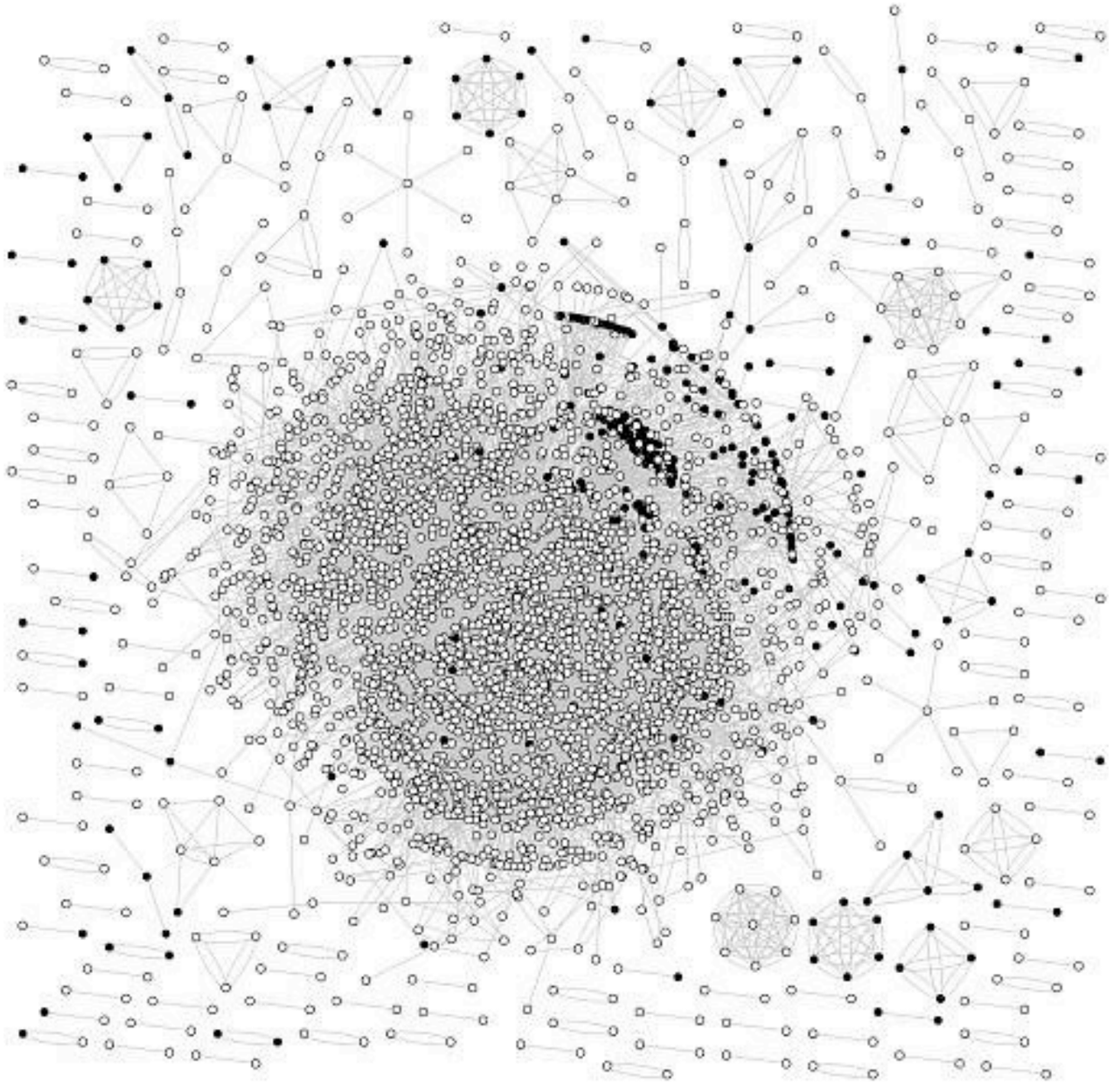

phenotype
Biomine
network
gene


phenotype
Biomine network
gene
What is the most relevant subnetwork (with at most k edges) connecting these?


phenotype
Biomine network
gene
What is the most relevant subnetwork (with at most k edges) connecting these?


phenotype
Biomine network
Should there be a link?
If so, of what type?
Why?
gene

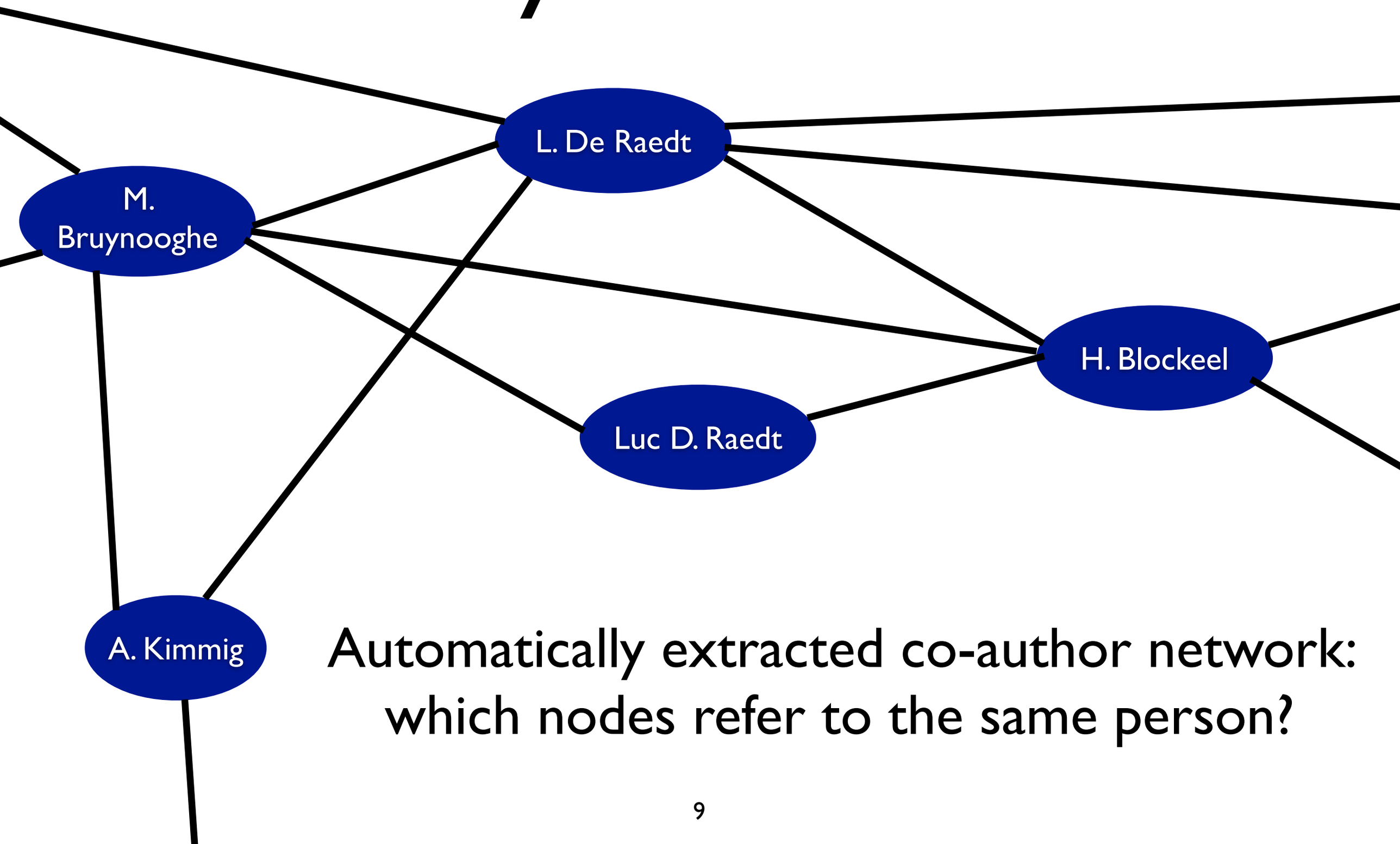
Node Classification



Can we predict the type of a node given information on its neighbors?

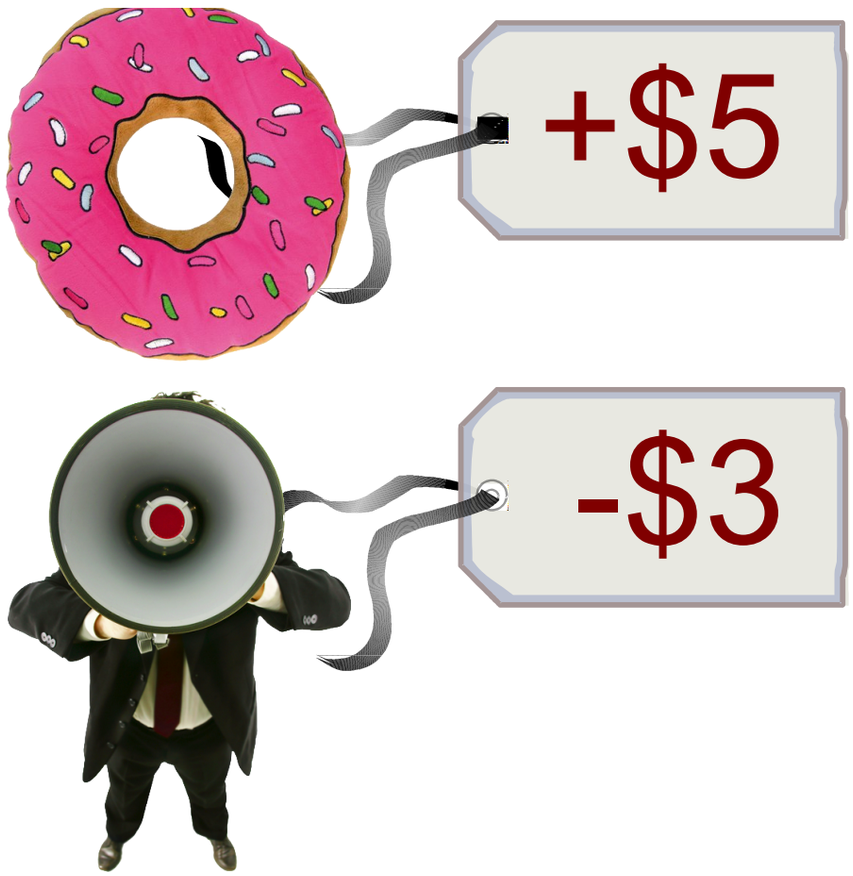
e.g., the type of a webpage given its links and the words on the page?

Entity Resolution

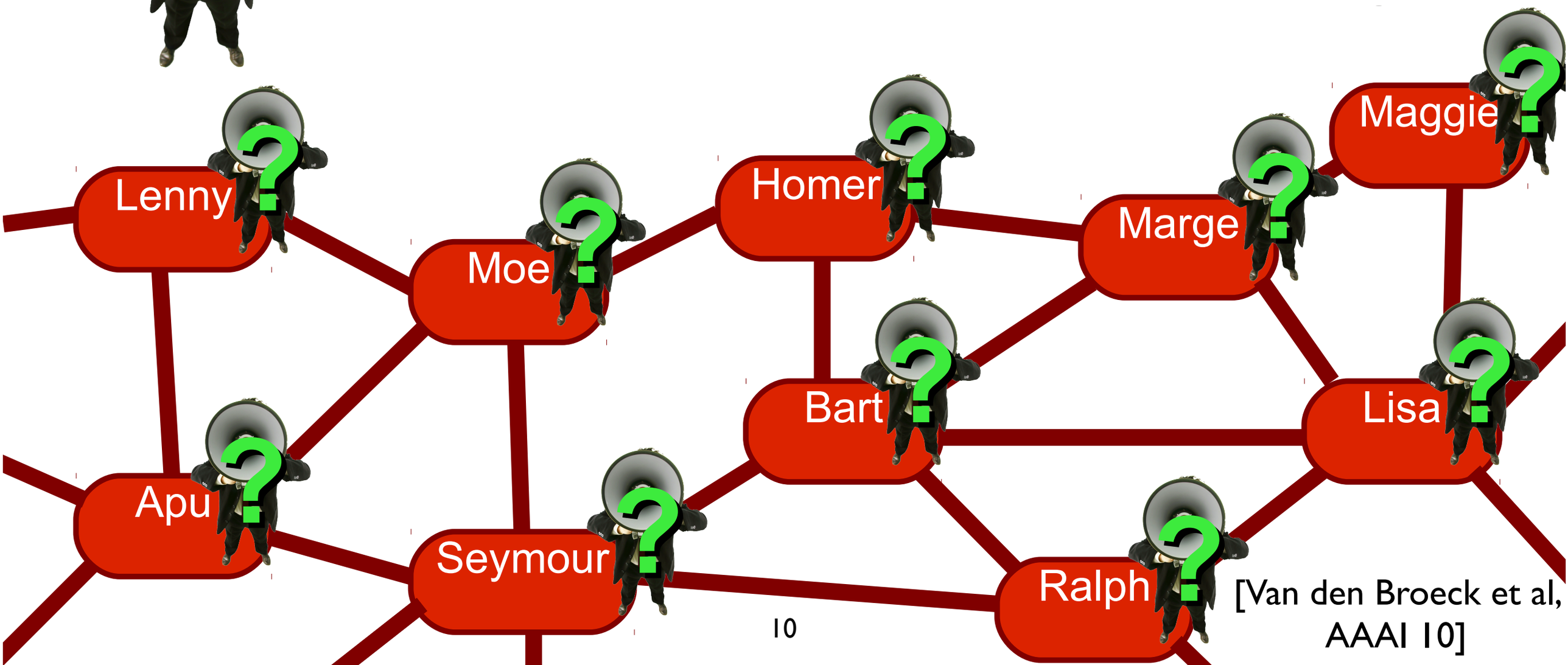


Automatically extracted co-author network: which nodes refer to the same person?

Viral Marketing

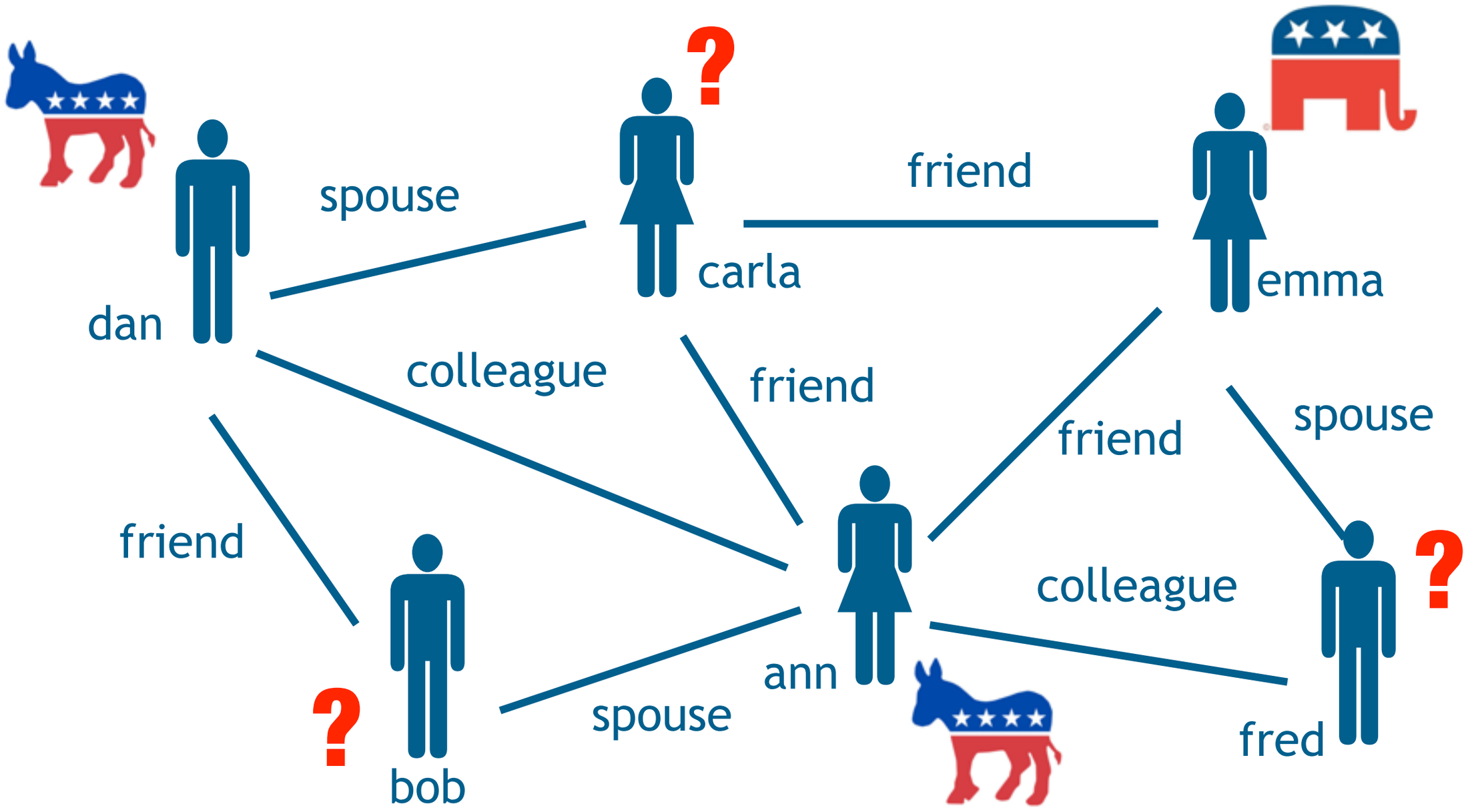


Which advertising strategy maximizes expected profit?



[Van den Broeck et al, AAAI 10]

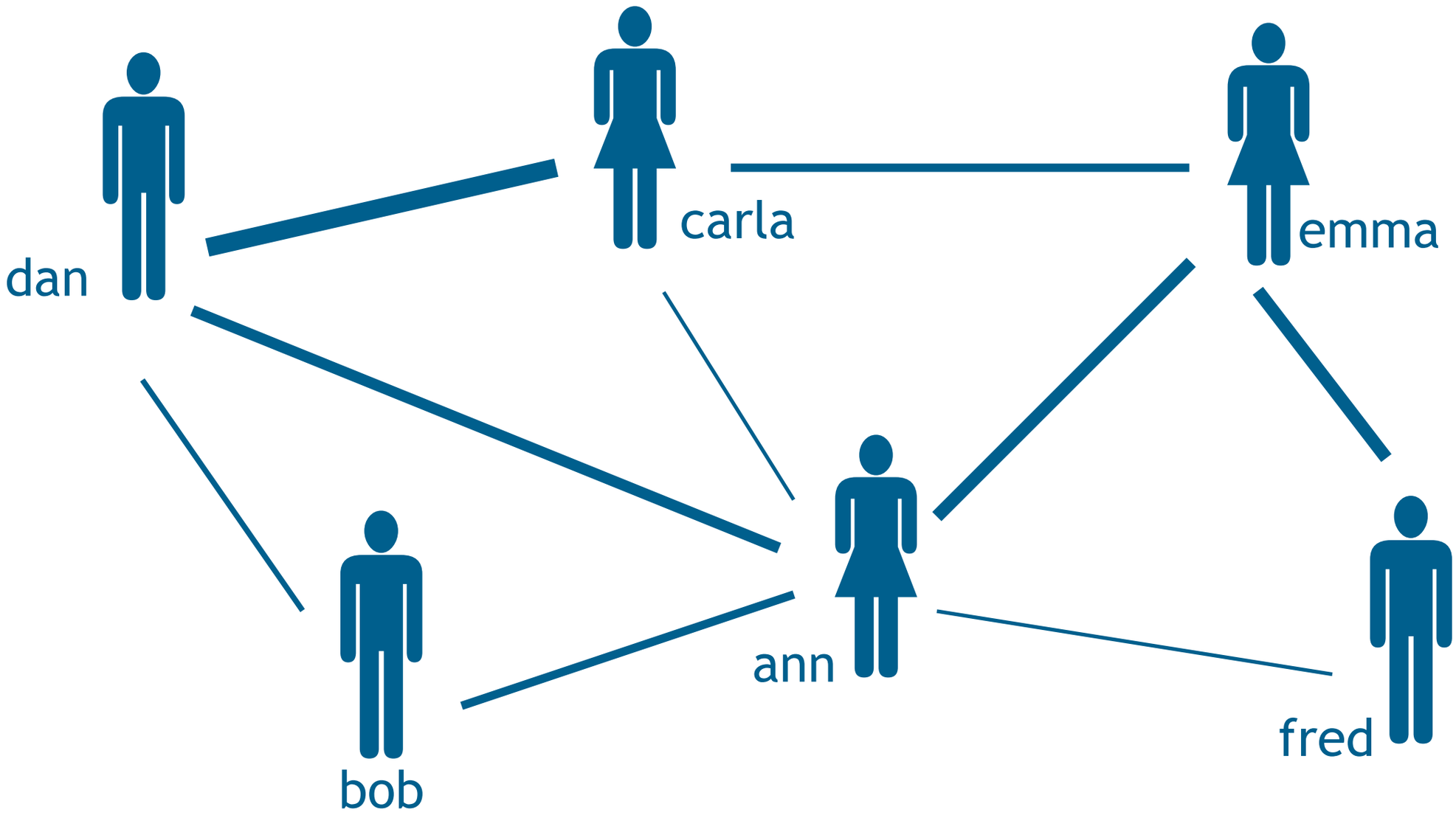
Voter Opinion Modeling



Can we predict preferences?

[Bach et al, NIPS 12]

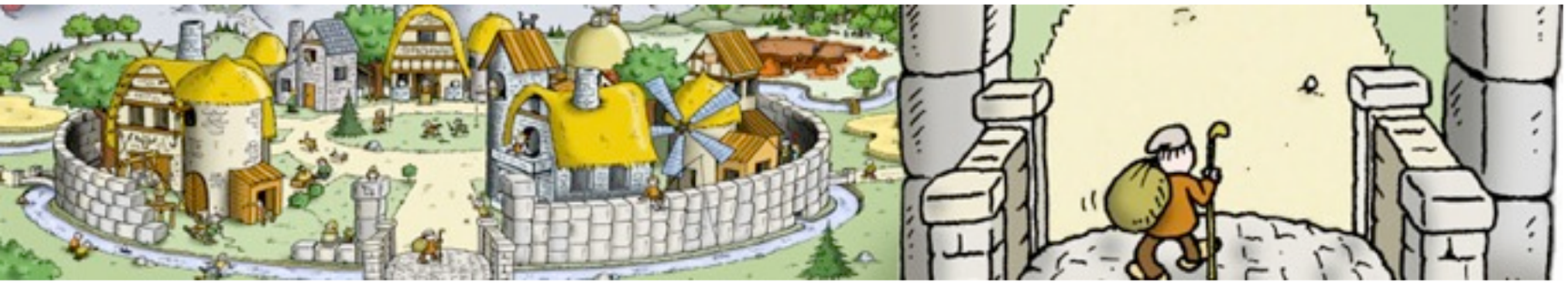
Social Trust



Can we predict strength of ties?

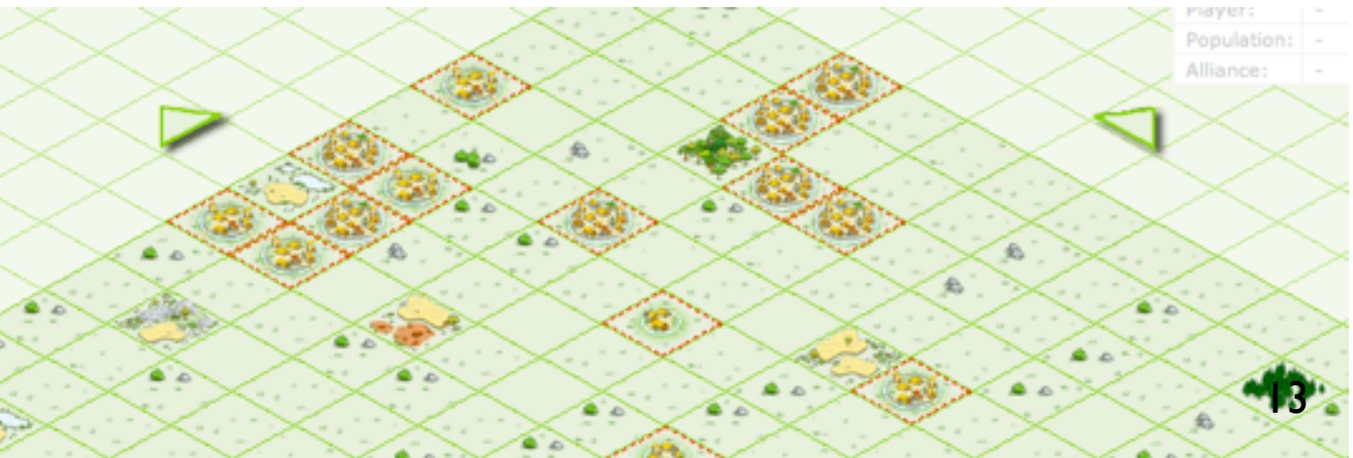
[Huang et al, SBP 13]

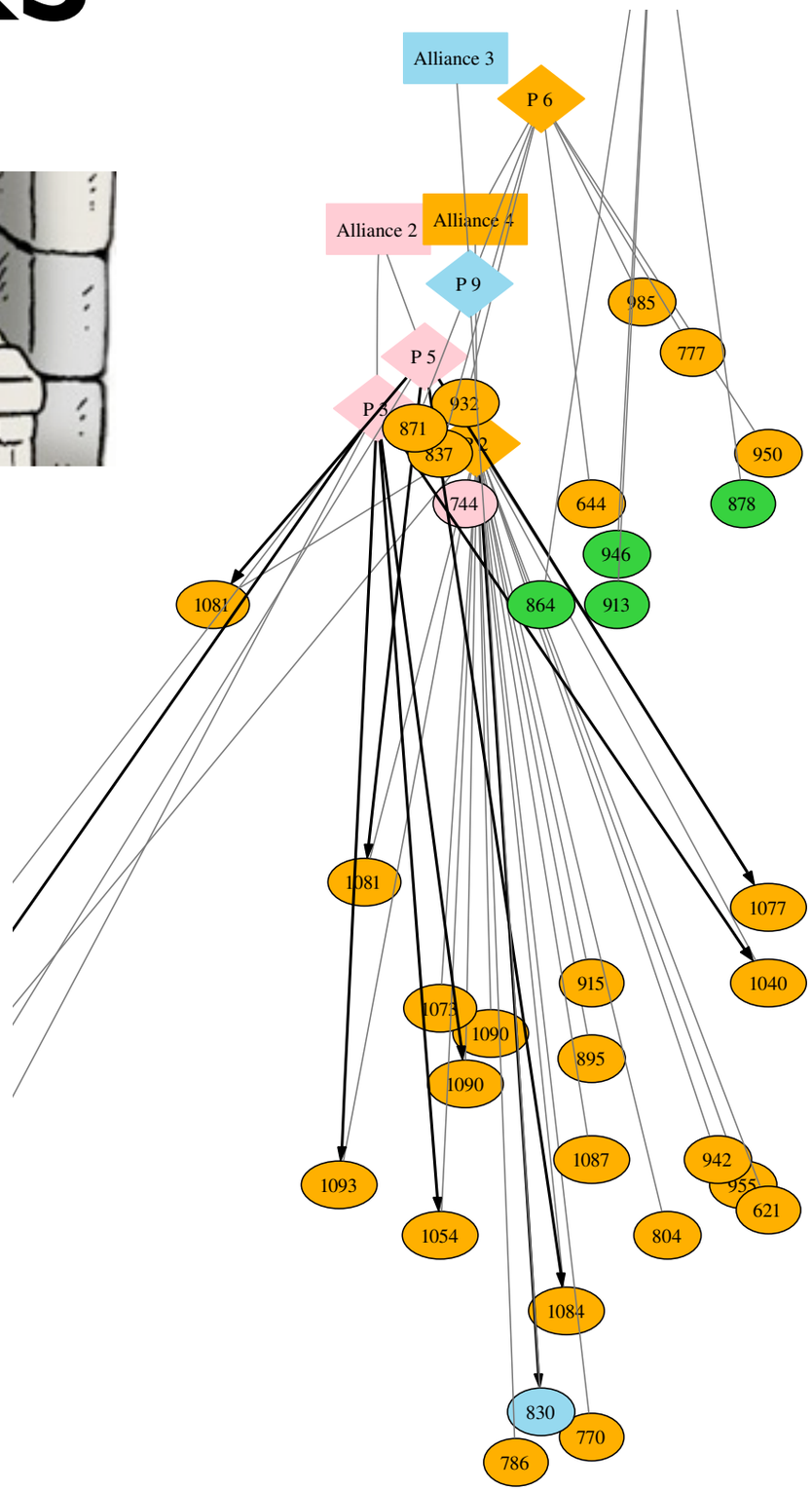
Dynamic networks



Travian: A massively multiplayer real-time strategy game

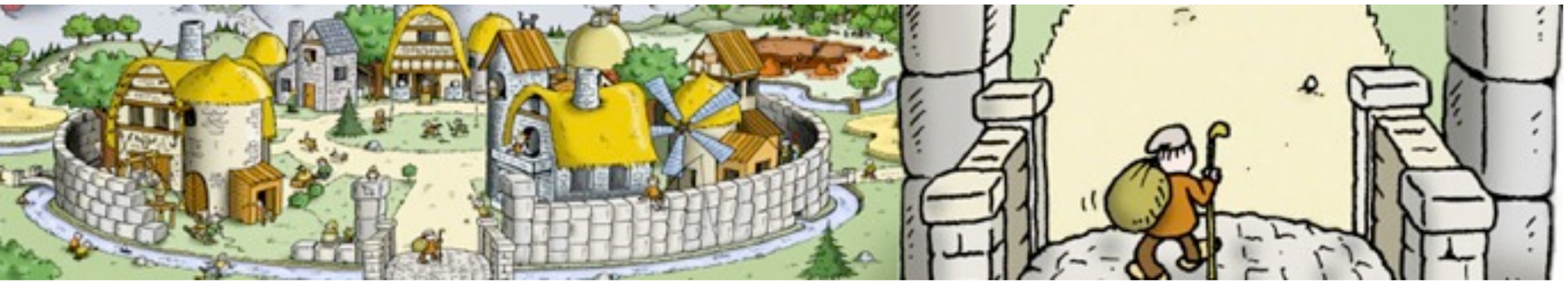
Can we build a model
of this world ?
Can we use it for playing
better ?





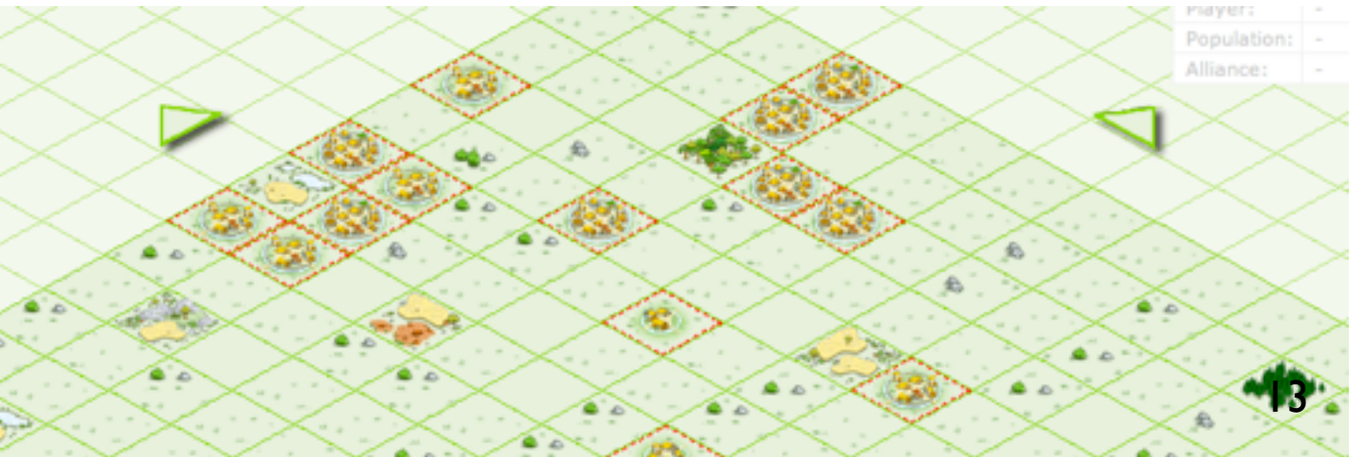
[Thon et al, MLJ 11]

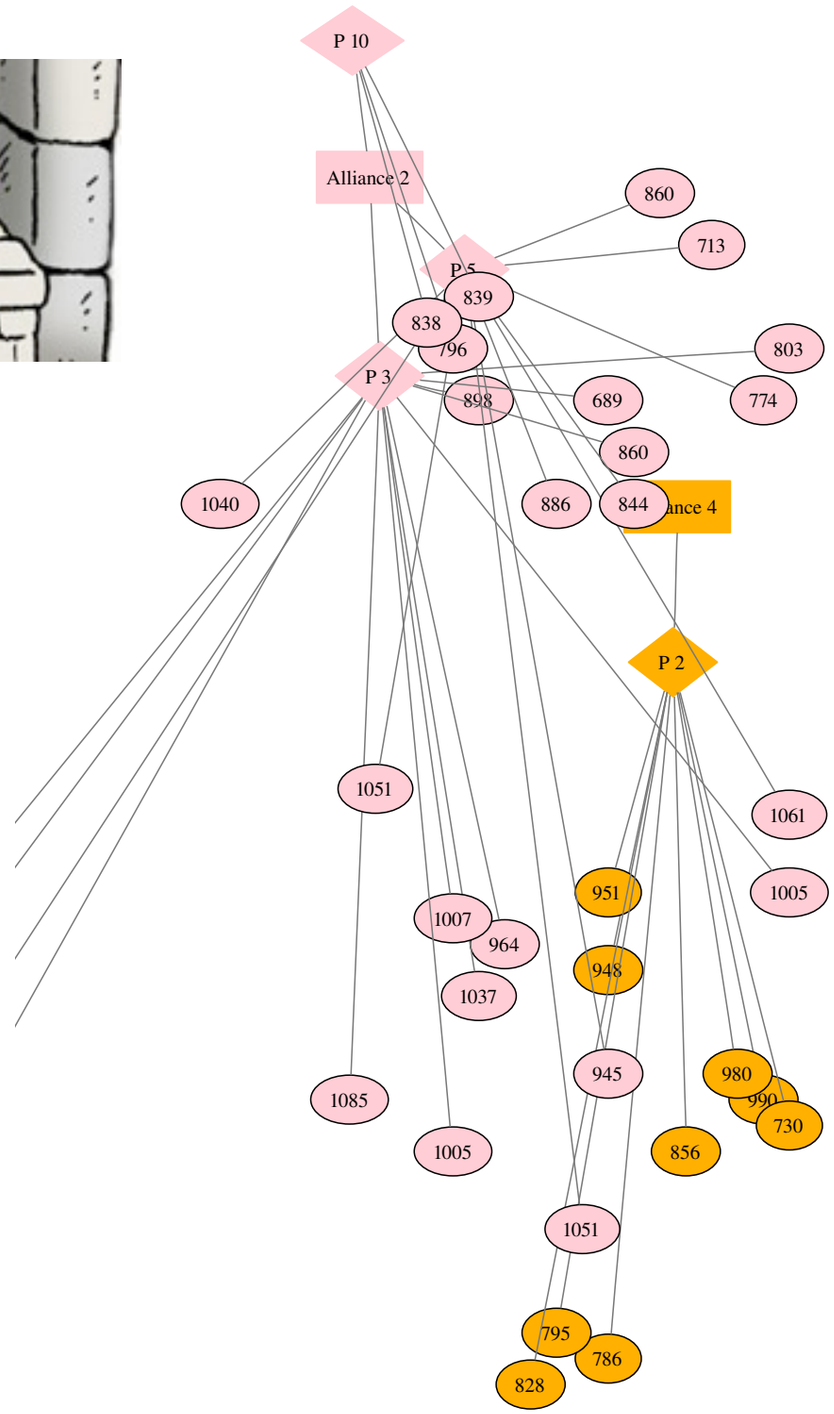
Dynamic networks



Travian: A massively multiplayer real-time strategy game

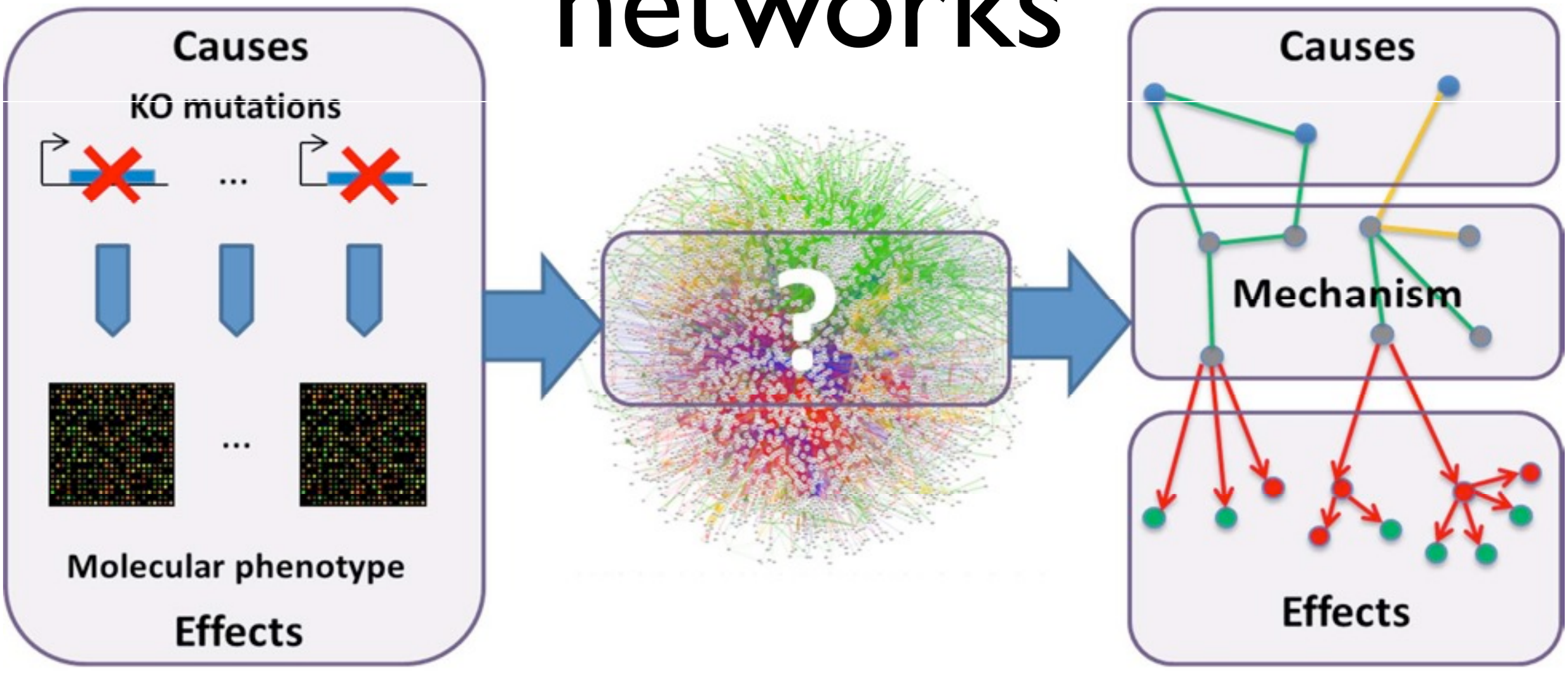
Can we build a model
of this world ?
Can we use it for playing
better ?





[Thon et al, MLJ 11]

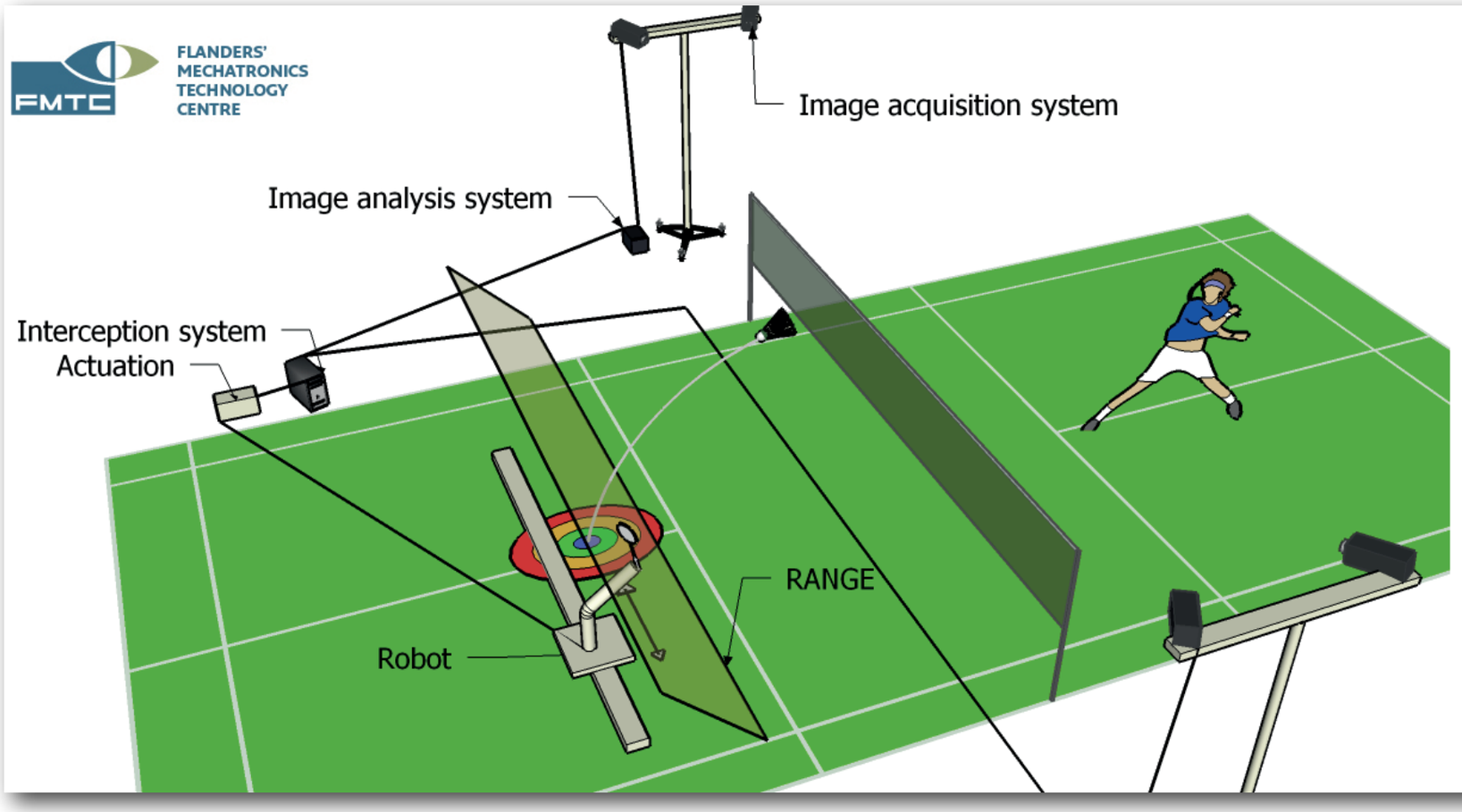
Molecular interaction networks

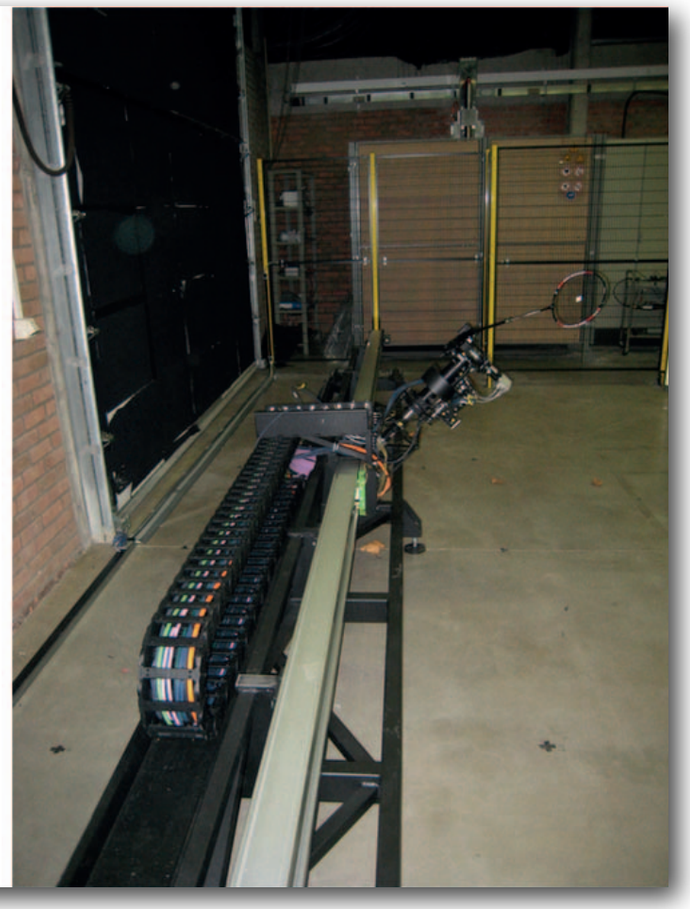


Can we find the mechanism connecting causes to effects?

[De Maeyer et al, Molecular Biosystems 13]

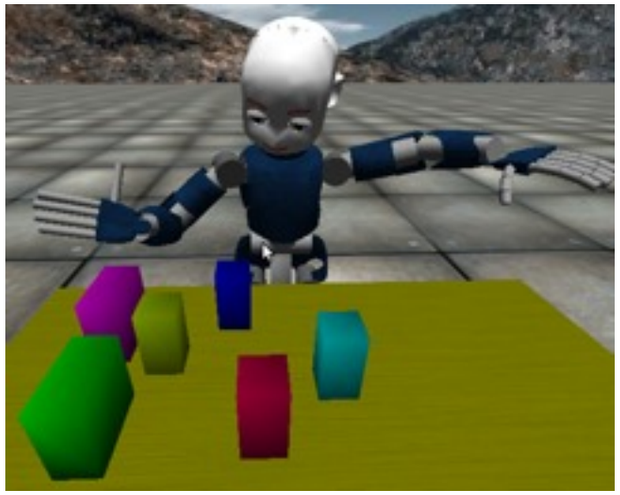
Diagnosing machine failures



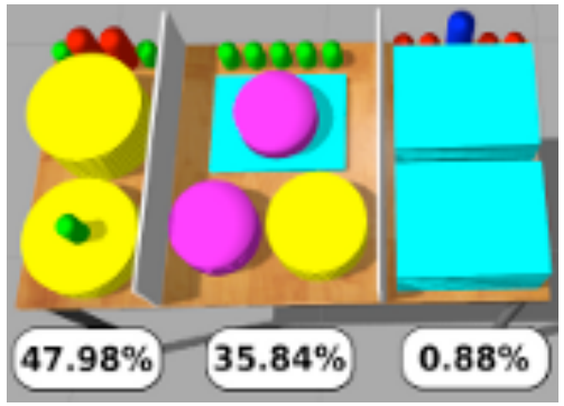


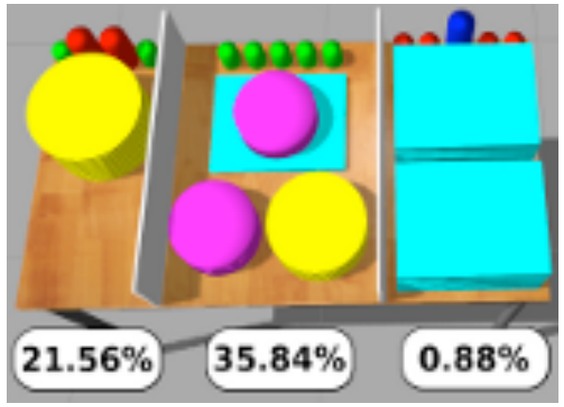
Can we build a model of the robot's working and use it to find causes of failures?

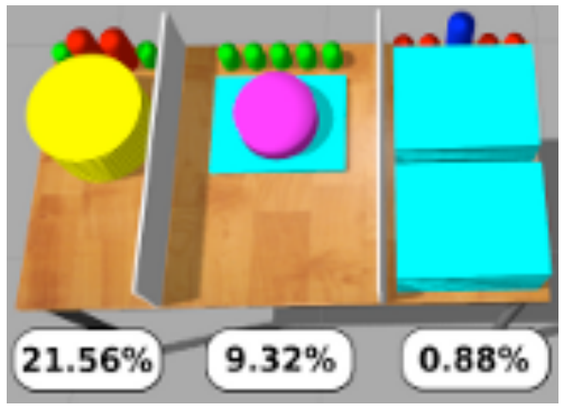
[Schramm, Meert and Driessens]



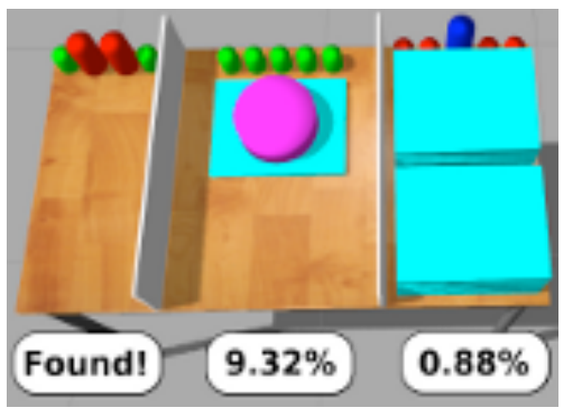
Robotics





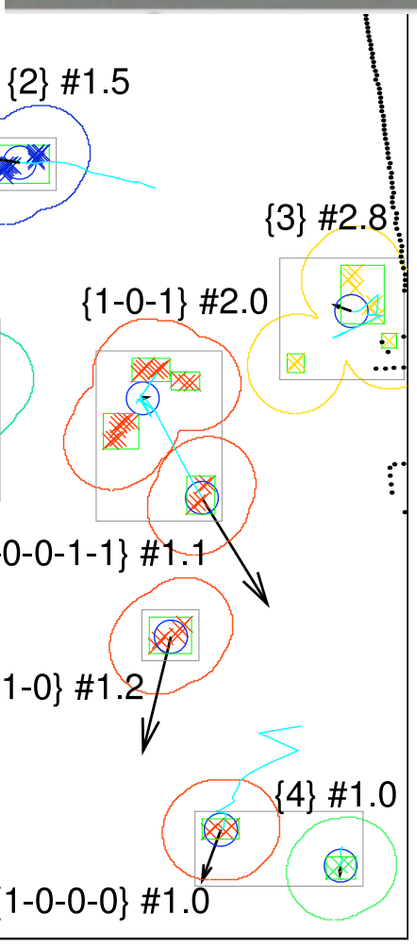


- How to achieve a specific configuration of objects on the shelf?
- Where's the orange mug?
- Where's something to serve soup in?



[Moldovan et al, ICRA 12, 14]

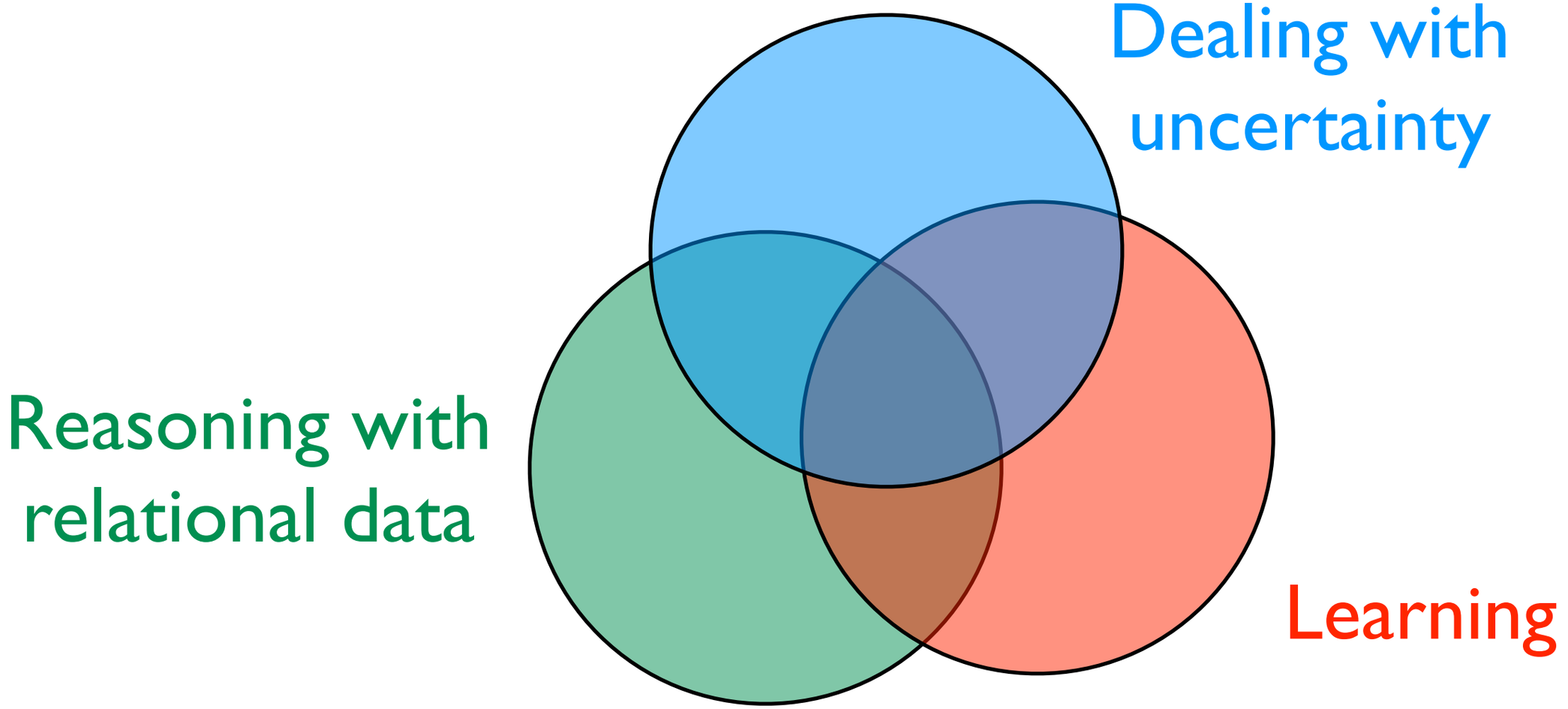
Analyzing Video Data



- Track people or objects over time? Even if temporarily hidden?
- Recognize activities?
- Infer object properties?

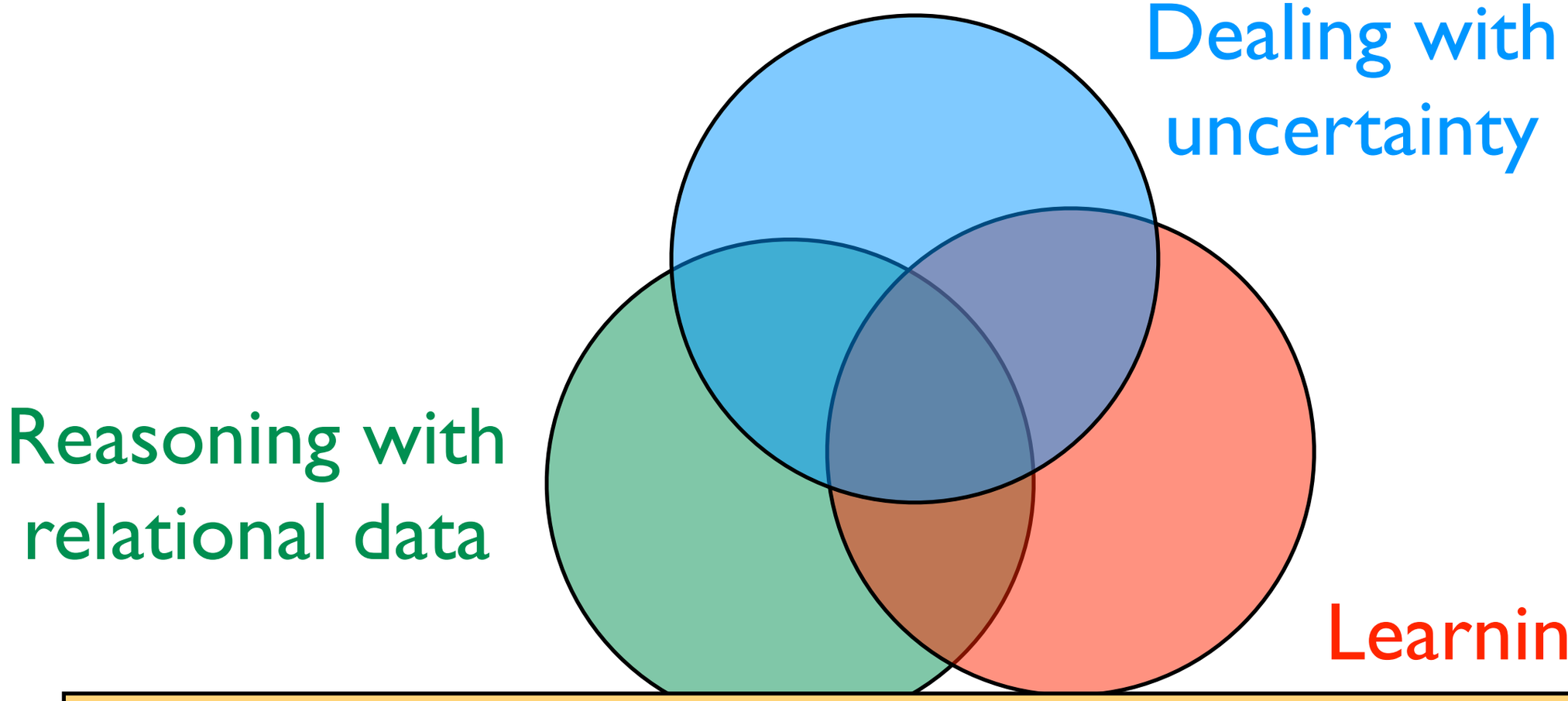
[Skarlatidis et al, TPLP 14;
Nitti et al, IROS 13, ICRA 14]

Common theme



Statistical relational learning, probabilistic logic learning, probabilistic programming, ...

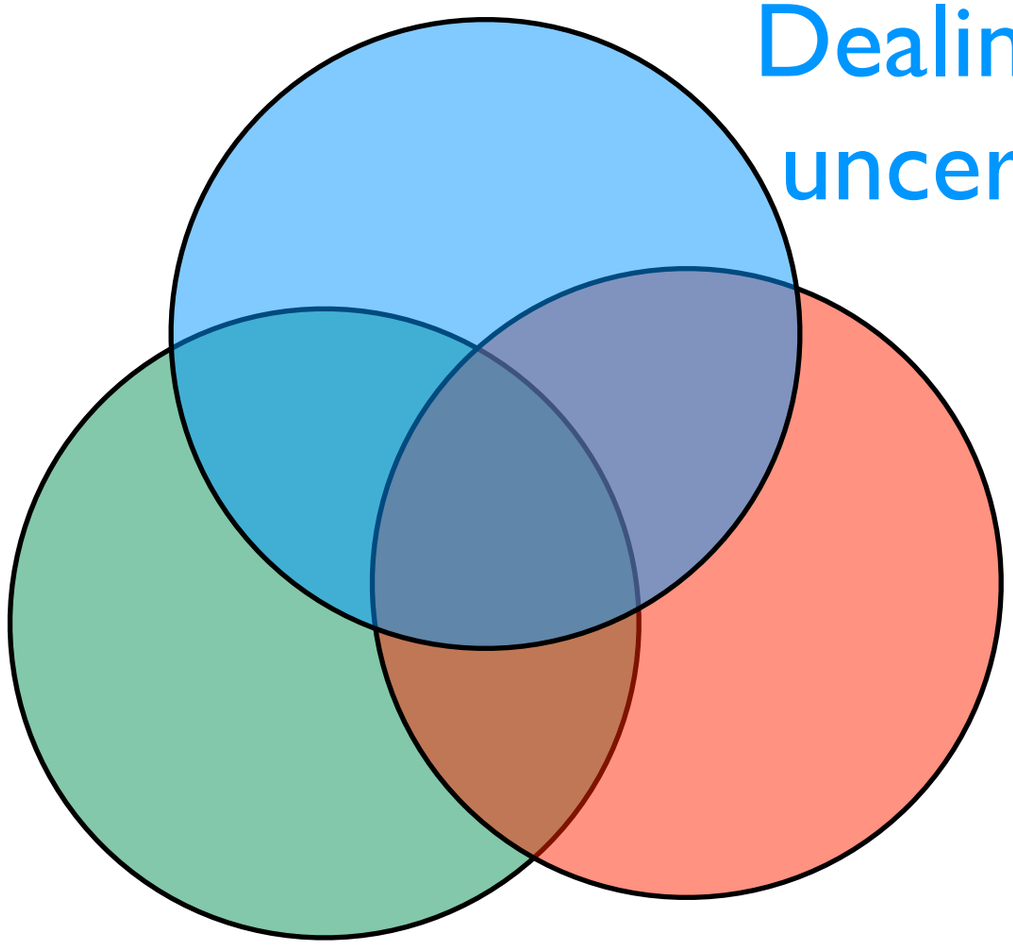
Common theme



- many different formalisms
- our focus: probabilistic logic programming
- won't cover lifted graphical models such as MLNs, PRMs, PSL, RBNs, ...

ProbLog

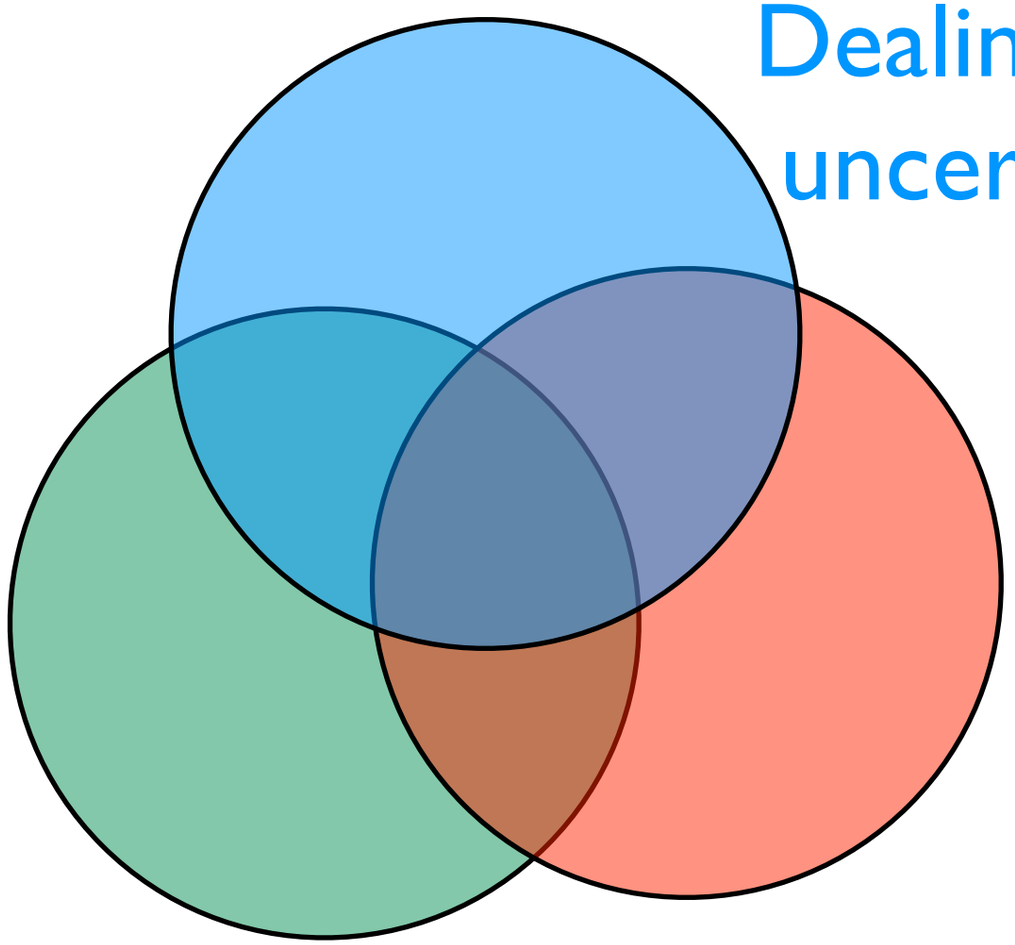
probabilistic Prolog



http://dtai.cs.kuleuven.be/problog/

ProbLog

probabilistic Prolog

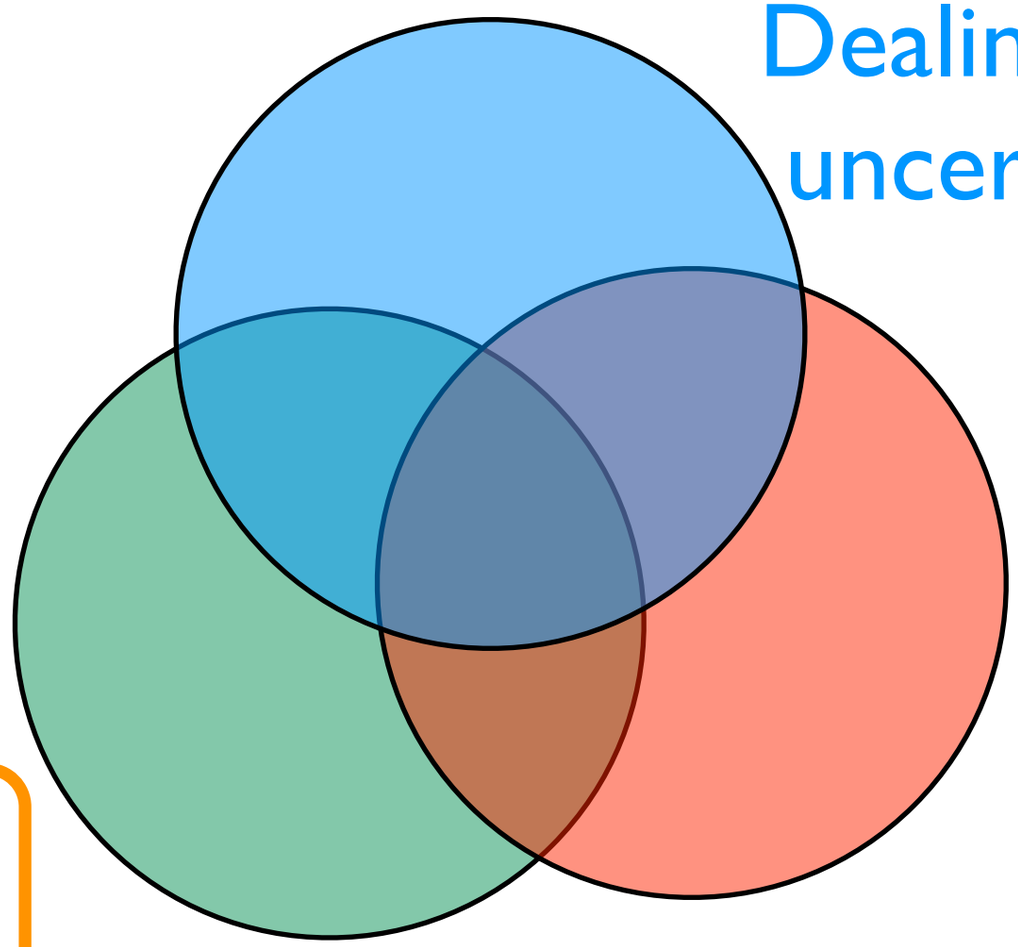



```
stress(ann).
influences(ann,bob).
influences(bob,carl).

smokes(X) :- stress(X).
smokes(X) :-
     influences(Y,X), smokes(Y).
```


http://dtai.cs.kuleuven.be/problog/

ProbLog

probabilistic Prolog




```
stress(ann).
influences(ann,bob).
influences(bob,carl).
```

one world

```
smokes(X) :- stress(X).
smokes(X) :-
     influences(Y,X), smokes(Y).
```

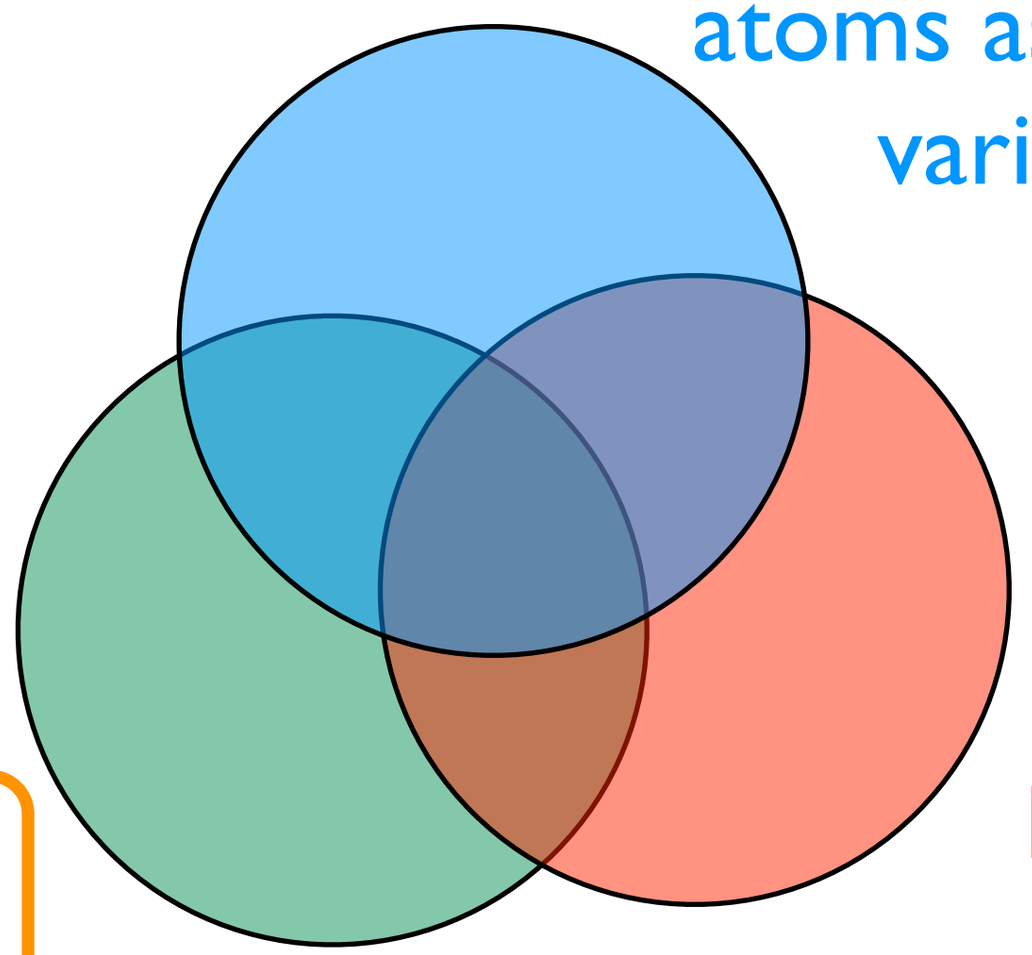

http://dtai.cs.kuleuven.be/problog/

ProbLog

probabilistic Prolog

```
0.8::stress(ann).
0.6::influences(ann,bob).
0.2::influences(bob,carl).
```

atoms as random variables



Prolog / logic programming

Learning

```
stress(ann).
influences(ann,bob).
influences(bob,carl).
```

one world

```
smokes(X) :- stress(X).
smokes(X) :-
     influences(Y,X), smokes(Y).
```


http://dtai.cs.kuleuven.be/problog/

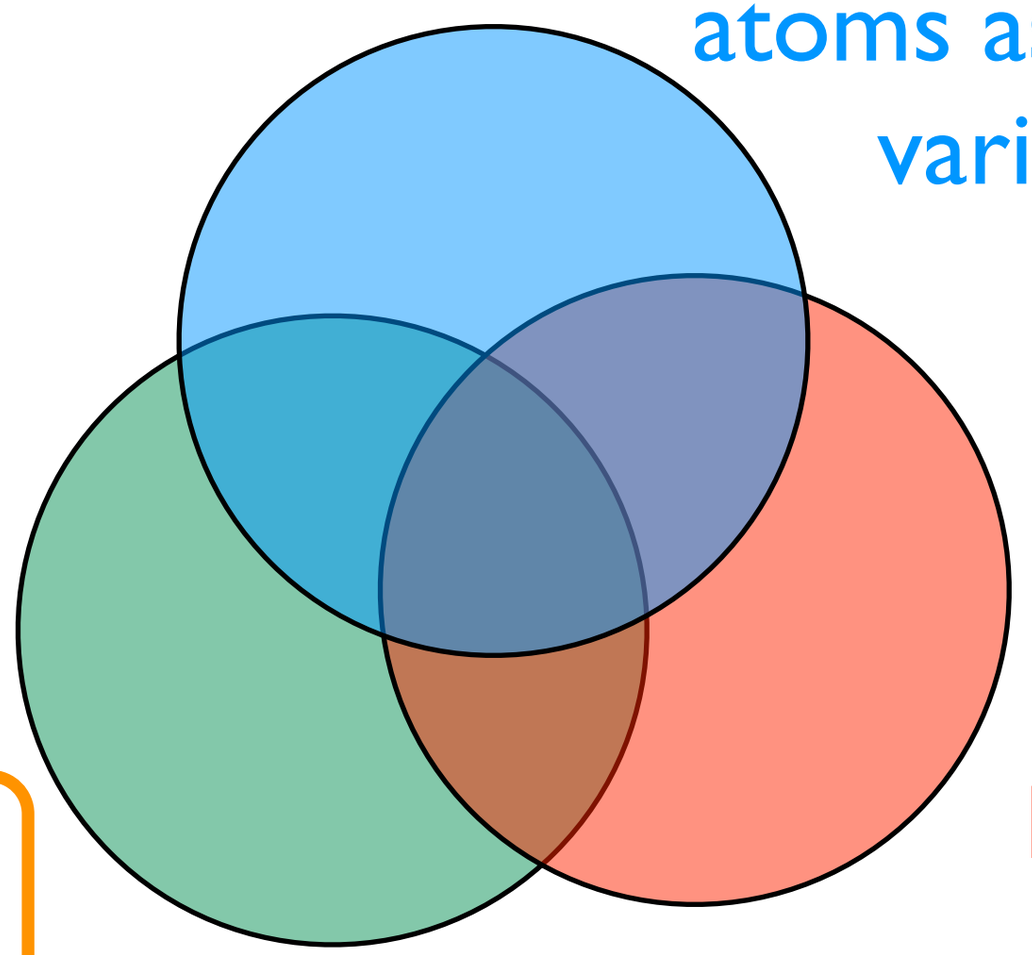
ProbLog

probabilistic Prolog

several possible worlds

```
0.8::stress(ann).
0.6::influences(ann,bob).
0.2::influences(bob,carl).
```

atoms as random variables



Prolog / logic programming

Learning

```
stress(ann).
influences(ann,bob).
influences(bob,carl).
```

one world

```
smokes(X) :- stress(X).
smokes(X) :-
     influences(Y,X), smokes(Y).
```


http://dtai.cs.kuleuven.be/problog/

ProbLog

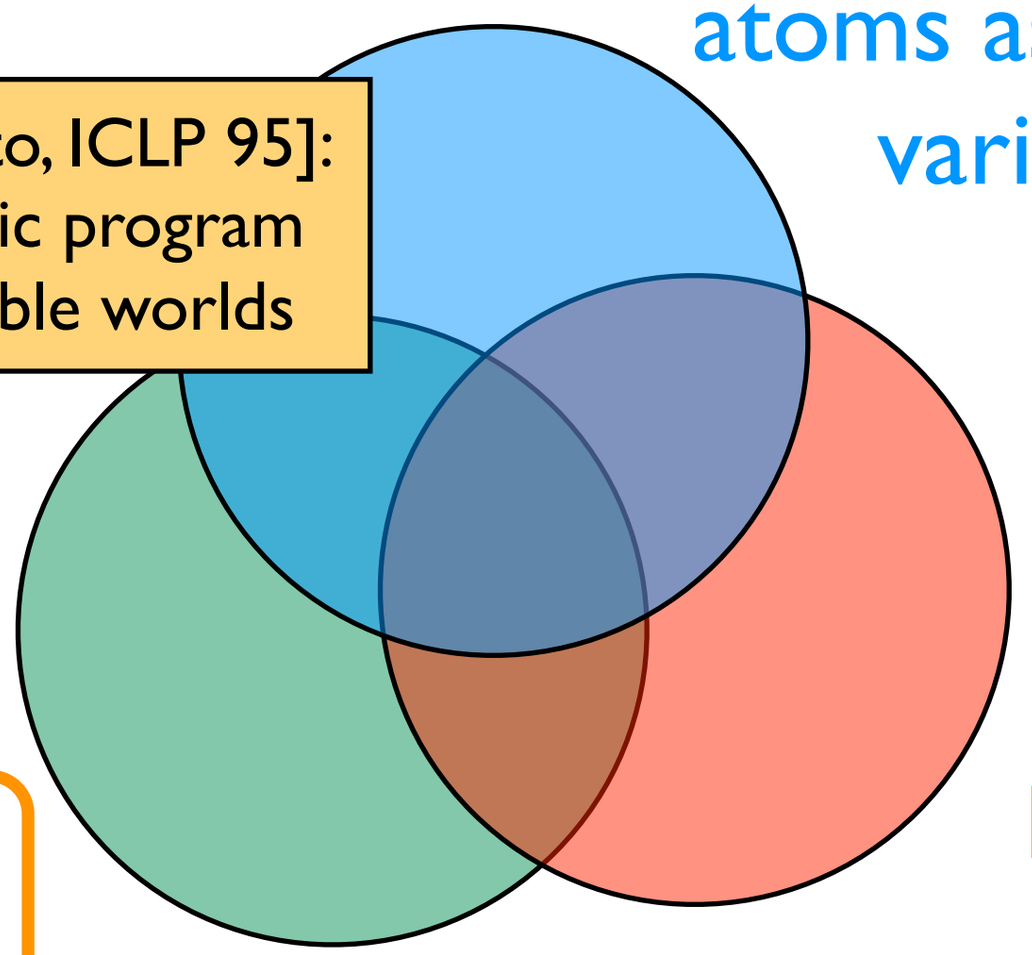
probabilistic Prolog

several possible worlds

```
0.8::stress(ann).
0.6::influences(ann,bob).
0.2::influences(bob,carl).
```

atoms as random variables

Distribution Semantics [Sato, ICLP 95]:
probabilistic choices + logic program
→ distribution over possible worlds



Prolog / logic programming

Learning

```
stress(ann).
influences(ann,bob).
influences(bob,carl).
```

one world

```
smokes(X) :- stress(X).
smokes(X) :-
     influences(Y,X), smokes(Y).
```


http://dtai.cs.kuleuven.be/problog/

ProbLog

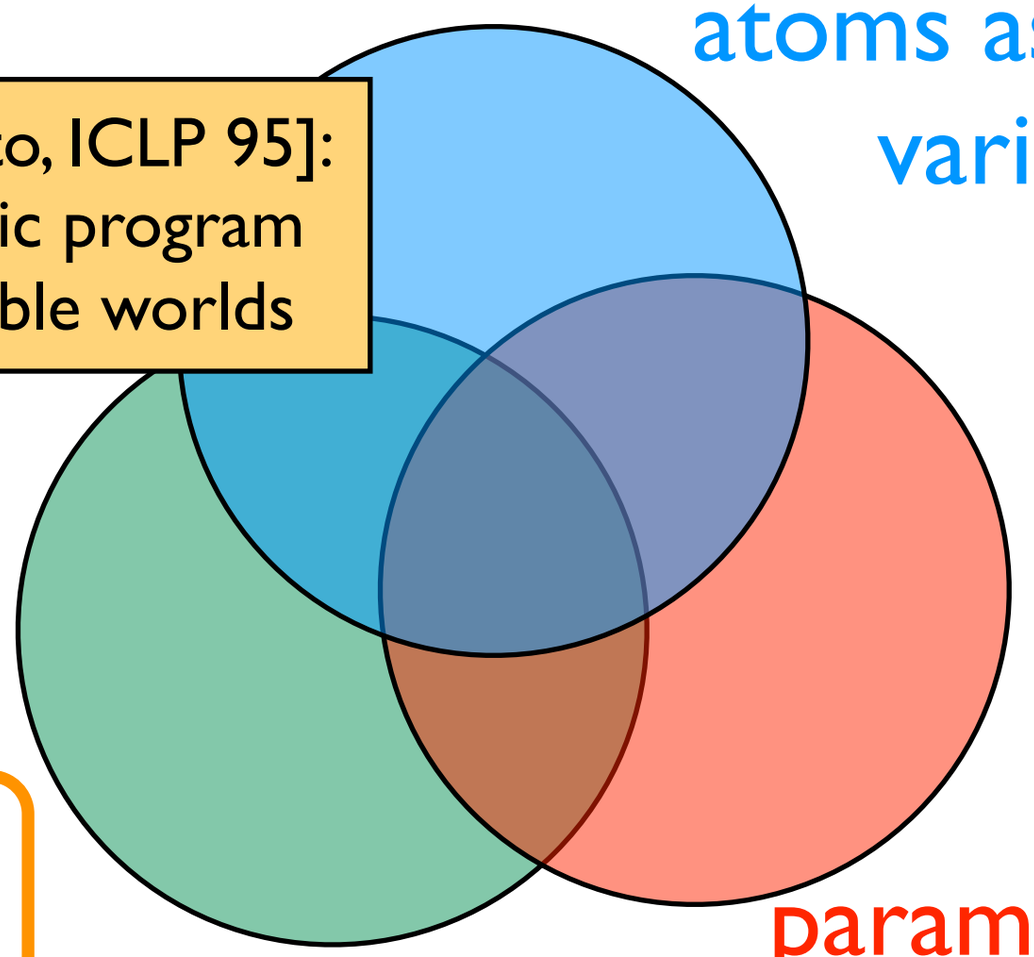
probabilistic Prolog

Distribution Semantics [Sato, ICLP 95]:
probabilistic choices + logic program
→ distribution over possible worlds

several possible worlds

```
0.8::stress(ann).
0.6::influences(ann,bob).
0.2::influences(bob,carl).
```

atoms as random variables



Prolog / logic programming

```
stress(ann).
influences(ann,bob).
influences(bob,carl).
```

one world

```
smokes(X) :- stress(X).
smokes(X) :-
     influences(Y,X), smokes(Y).
```

parameter learning, adapted relational learning techniques

http://dtai.cs.kuleuven.be/problog/

Probabilistic Prologs: Two Views

- Distribution semantics:
  - probability distribution over interpretations
  - degree of belief
- Stochastic Logic Programs (SLPs):
  - probability distribution over query answers
  - like in probabilistic grammars

Probabilistic Prologs: Two Views

- Distribution semantics:
  - probability distribution over interpretations
  - degree of belief
- Stochastic Logic Programs (SLPs):
  - probability distribution over query answers
  - like in probabilistic grammars

Probabilistic Logic Programming

Distribution Semantics [Sato, ICLP 95]:
probabilistic choices + logic program
→ distribution over possible worlds

e.g., PRISM, ICL, ProbLog, LPADs, CP-logic, ...

multi-valued switches

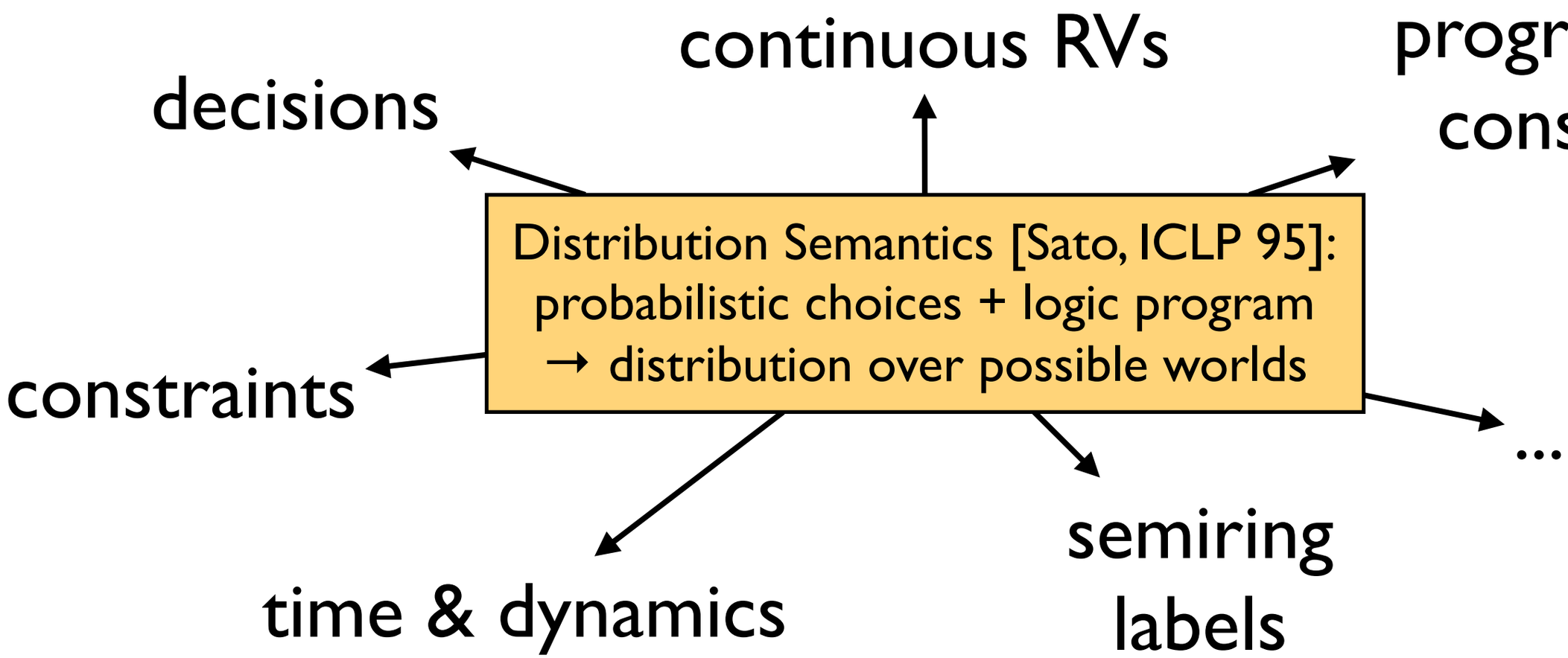
probabilistic alternatives

probabilistic facts

annotated disjunctions

causal-probabilistic laws

Extensions of basic PLP

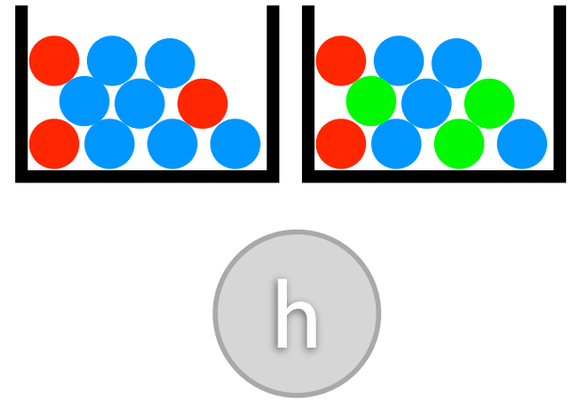


Roadmap

- Modeling (with detours to related work)
- Reasoning (and a bit of learning)
- Language extensions

ProbLog by example:

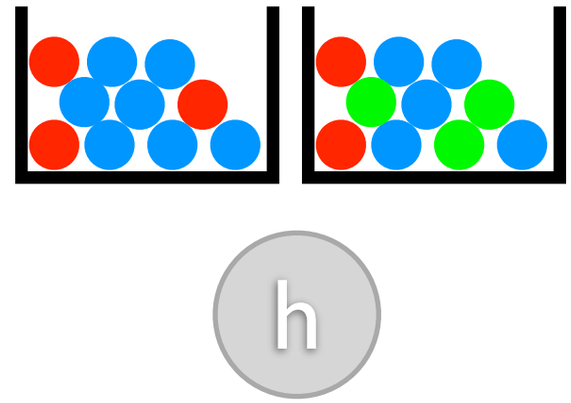
A bit of gambling



- toss (biased) coin & draw ball from each urn
- win if (heads and a red ball) or (two balls of same color)

ProbLog by example:

A bit of gambling



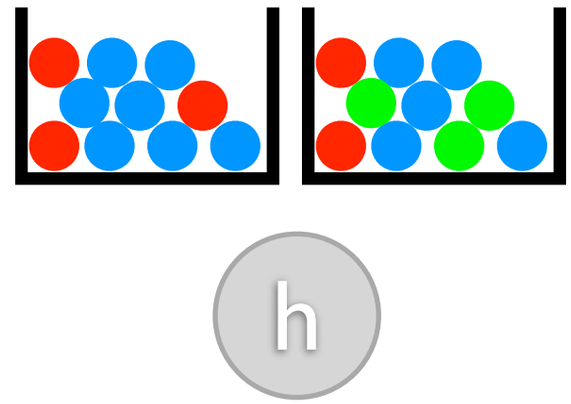
- toss (biased) coin & draw ball from each urn
- win if (heads and a red ball) or (two balls of same color)

```
0.4 :: heads.
```

probabilistic fact: heads is true with probability 0.4 (and false with 0.6)

ProbLog by example:

A bit of gambling



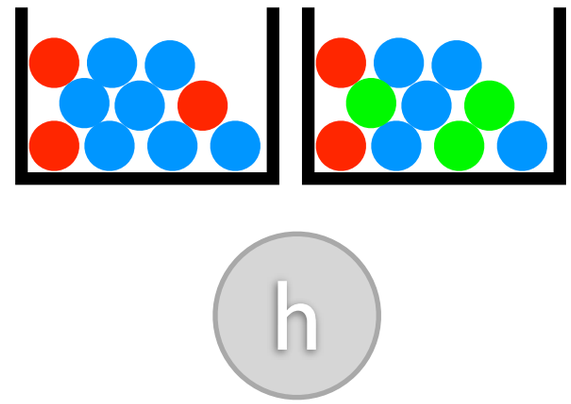
- toss (biased) coin & draw ball from each urn
- win if (heads and a red ball) or (two balls of same color)

```
0.4 :: heads.
0.3 :: col(1,red); 0.7 :: col(1,blue) <- true.
```

annotated disjunction: first ball is red with probability 0.3 and blue with 0.7

ProbLog by example:

A bit of gambling



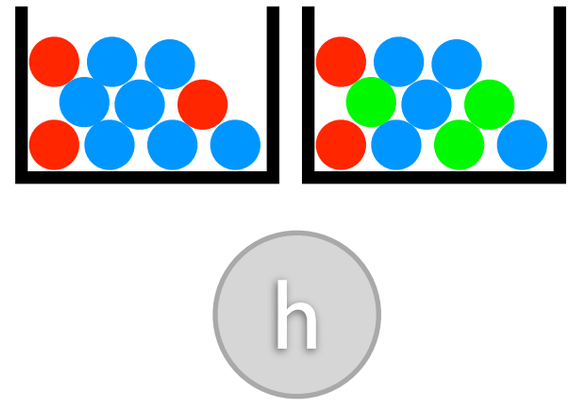
- toss (biased) coin & draw ball from each urn
- win if (heads and a red ball) or (two balls of same color)

```
0.4 :: heads.

0.3 :: col(1,red); 0.7 :: col(1,blue) <- true.
0.2 :: col(2,red); 0.3 :: col(2,green);
                   0.5 :: col(2,blue) <- true.
```

annotated disjunction: second ball is red with probability 0.2, green with 0.3, and blue with 0.5

ProbLog by example:

A bit of gambling



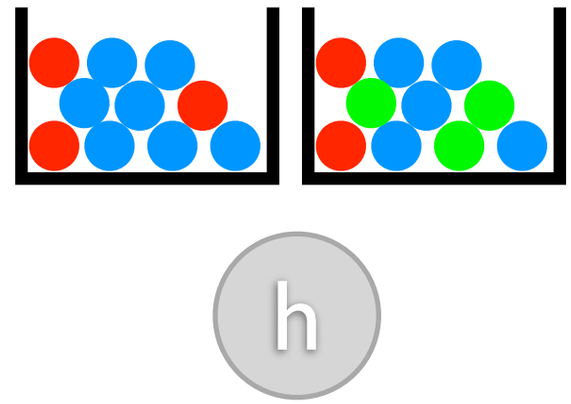
- toss (biased) coin & draw ball from each urn
- win if (heads and a red ball) or (two balls of same color)

```
0.4 :: heads.

0.3 :: col(1,red); 0.7 :: col(1,blue) <- true.
0.2 :: col(2,red); 0.3 :: col(2,green);
                   0.5 :: col(2,blue) <- true.

win :- heads, col(_,red).
```

logical rule encoding background knowledge

ProbLog by example:

A bit of gambling



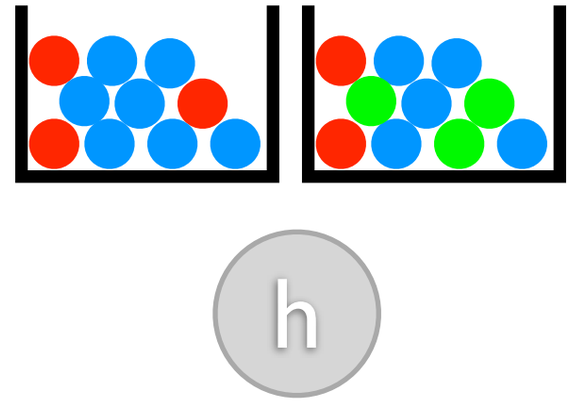
- toss (biased) coin & draw ball from each urn
- win if (heads and a red ball) or (two balls of same color)

```
0.4 :: heads.

0.3 :: col(1,red); 0.7 :: col(1,blue) <- true.
0.2 :: col(2,red); 0.3 :: col(2,green);
                   0.5 :: col(2,blue) <- true.

win :- heads, col(_,red).
win :- col(1,C), col(2,C).
```

logical rule encoding background knowledge

ProbLog by example:

A bit of gambling



- toss (biased) coin & draw ball from each urn
- win if (heads and a red ball) or (two balls of same color)

```
0.4 :: heads.

0.3 :: col(1,red); 0.7 :: col(1,blue) <- true.
0.2 :: col(2,red); 0.3 :: col(2,green);
                   0.5 :: col(2,blue) <- true.
```

probabilistic choices

```
win :- heads, col(_,red).
win :- col(1,C), col(2,C).
```

consequences

Questions

```
0.4 :: heads.

0.3 :: col(1,red); 0.7 :: col(1,blue) <- true.
0.2 :: col(2,red); 0.3 :: col(2,green); 0.5 :: col(2,blue) <- true.

win :- heads, col(_,red).
win :- col(1,C), col(2,C).
```

- Probability of **`win`**?
- Probability of **`win`** given **`col(2,green)`**?
- Most probable world where **`win`** is true?

Questions

```
0.4 :: heads.

0.3 :: col(1,red); 0.7 :: col(1,blue) <- true.
0.2 :: col(2,red); 0.3 :: col(2,green); 0.5 :: col(2,blue) <- true.

win :- heads, col(_,red).
win :- col(1,C), col(2,C).
```

marginal probability

- Probability of `win`?

query

- Probability of `win` given `col(2,green)`?
- Most probable world where `win` is true?

Questions

```
0.4 :: heads.

0.3 :: col(1,red); 0.7 :: col(1,blue) <- true.
0.2 :: col(2,red); 0.3 :: col(2,green); 0.5 :: col(2,blue) <- true.

win :- heads, col(_,red).
win :- col(1,C), col(2,C).
```

marginal probability

- Probability of **`win`**?

conditional probability

- Probability of **`win`** given **`col(2,green)`**?

evidence

- Most probable world where **`win`** is true?

Questions

```
0.4 :: heads.

0.3 :: col(1,red); 0.7 :: col(1,blue) <- true.
0.2 :: col(2,red); 0.3 :: col(2,green); 0.5 :: col(2,blue) <- true.

win :- heads, col(_,red).
win :- col(1,C), col(2,C).
```

marginal probability

- Probability of **`win`**?

conditional probability

- Probability of **`win`** given **`col(2,green)`**?

- Most probable world where **`win`** is true?

MPE inference

Possible Worlds

```
0.4 :: heads.

0.3 :: col(1,red); 0.7 :: col(1,blue) <- true.
0.2 :: col(2,red); 0.3 :: col(2,green); 0.5 :: col(2,blue) <- true.

win :- heads, col( ,red).
win :- col(1,C), col(2,C).
```

Possible Worlds

```
0.4 :: heads.

0.3 :: col(1,red); 0.7 :: col(1,blue) <- true.
0.2 :: col(2,red); 0.3 :: col(2,green); 0.5 :: col(2,blue) <- true.

win :- heads, col( ,red).
win :- col(1,C), col(2,C).
```

Possible Worlds

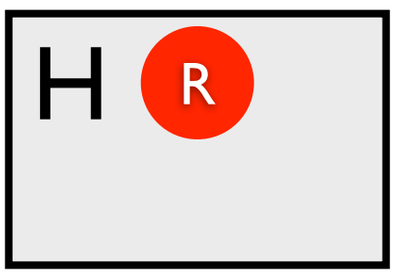
```
0.4 :: heads.

0.3 :: col(1,red); 0.7 :: col(1,blue) <- true.
0.2 :: col(2,red); 0.3 :: col(2,green); 0.5 :: col(2,blue) <- true.

win :- heads, col( ,red).
win :- col(1,C), col(2,C).
```

0.4

Possible Worlds

```
0.4 :: heads.

0.3 :: col(1,red); 0.7 :: col(1,blue) <- true.
0.2 :: col(2,red); 0.3 :: col(2,green); 0.5 :: col(2,blue) <- true.

win :- heads, col(_,red).
win :- col(1,C), col(2,C).
```

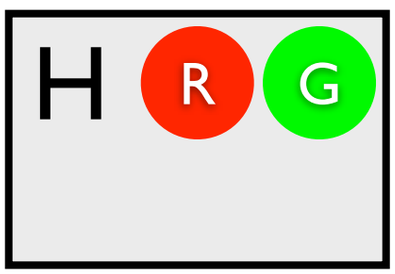
0.4 ×0.3



Possible Worlds

```
0.4 :: heads.

0.3 :: col(1,red); 0.7 :: col(1,blue) <- true.
0.2 :: col(2,red); 0.3 :: col(2,green); 0.5 :: col(2,blue) <- true.

win :- heads, col(_,red).
win :- col(1,C), col(2,C).
```

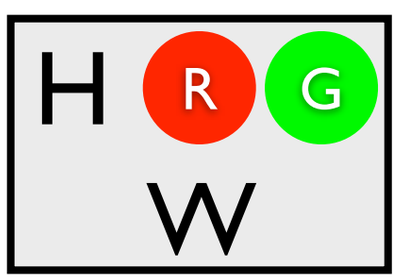
0.4 ×0.3 ×0.3



Possible Worlds

```
0.4 :: heads.

0.3 :: col(1,red); 0.7 :: col(1,blue) <- true.
0.2 :: col(2,red); 0.3 :: col(2,green); 0.5 :: col(2,blue) <- true.

win :- heads, col(_,red).
win :- col(1,C), col(2,C).
```

0.4 ×0.3 ×0.3



Possible Worlds

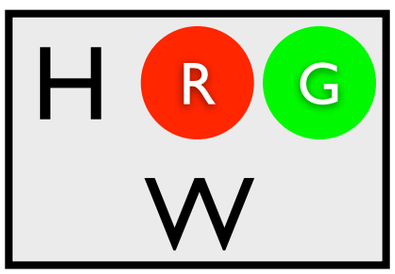
```
0.4 :: heads.

0.3 :: col(1,red); 0.7 :: col(1,blue) <- true.
0.2 :: col(2,red); 0.3 :: col(2,green); 0.5 :: col(2,blue) <- true.

win :- heads, col( ,red).
win :- col(1,C), col(2,C).
```

$0.4 \times 0.3 \times 0.3$ $(1-0.4)$

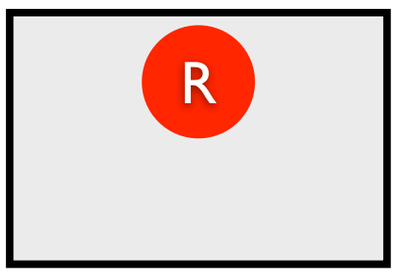
Possible Worlds

```
0.4 :: heads.

0.3 :: col(1,red); 0.7 :: col(1,blue) <- true.
0.2 :: col(2,red); 0.3 :: col(2,green); 0.5 :: col(2,blue) <- true.

win :- heads, col(_,red).
win :- col(1,C), col(2,C).
```

$0.4 \times 0.3 \times 0.3$



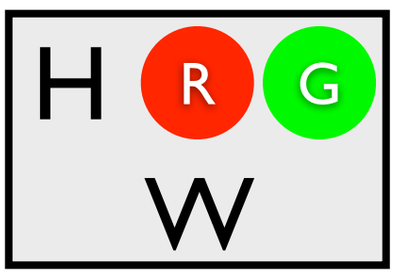
$(1-0.4) \times 0.3$



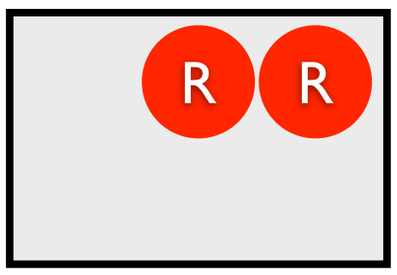
Possible Worlds

```
0.4 :: heads.

0.3 :: col(1,red); 0.7 :: col(1,blue) <- true.
0.2 :: col(2,red); 0.3 :: col(2,green); 0.5 :: col(2,blue) <- true.

win :- heads, col(_,red).
win :- col(1,C), col(2,C).
```

$0.4 \times 0.3 \times 0.3$



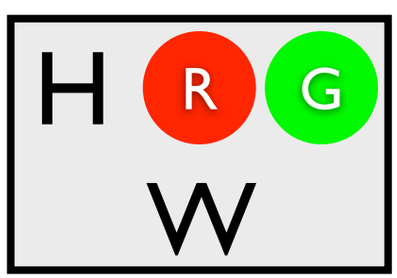
$(1-0.4) \times 0.3 \times 0.2$



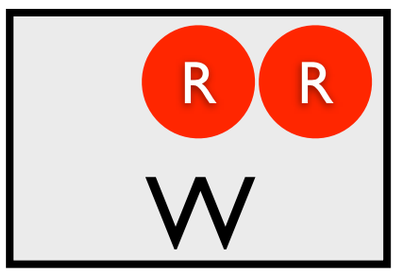
Possible Worlds

```
0.4 :: heads.

0.3 :: col(1,red); 0.7 :: col(1,blue) <- true.
0.2 :: col(2,red); 0.3 :: col(2,green); 0.5 :: col(2,blue) <- true.

win :- heads, col(_,red).
win :- col(1,C), col(2,C).
```

$0.4 \times 0.3 \times 0.3$



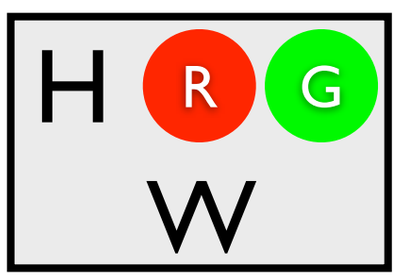
$(1-0.4) \times 0.3 \times 0.2$



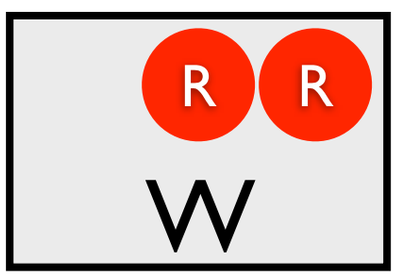
Possible Worlds

```
0.4 :: heads.

0.3 :: col(1,red); 0.7 :: col(1,blue) <- true.
0.2 :: col(2,red); 0.3 :: col(2,green); 0.5 :: col(2,blue) <- true.

win :- heads, col( ,red).
win :- col(1,C), col(2,C).
```

$0.4 \times 0.3 \times 0.3$



$(1-0.4) \times 0.3 \times 0.2$

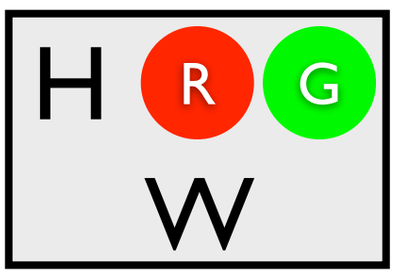


$(1-0.4)$

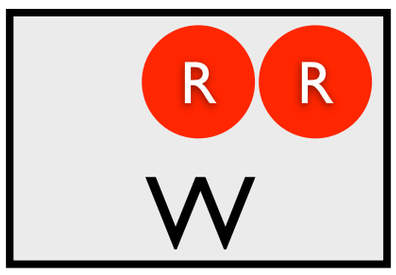
Possible Worlds

```
0.4 :: heads.

0.3 :: col(1,red); 0.7 :: col(1,blue) <- true.
0.2 :: col(2,red); 0.3 :: col(2,green); 0.5 :: col(2,blue) <- true.

win :- heads, col( ,red).
win :- col(1,C), col(2,C).
```

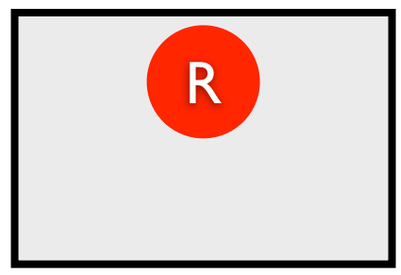
0.4 ×0.3 ×0.3



(1−0.4)×0.3 ×0.2



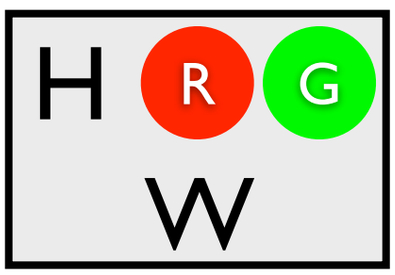
(1−0.4)×0.3



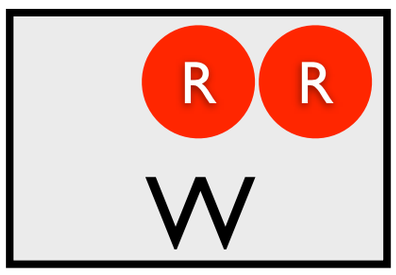
Possible Worlds

```
0.4 :: heads.

0.3 :: col(1,red); 0.7 :: col(1,blue) <- true.
0.2 :: col(2,red); 0.3 :: col(2,green); 0.5 :: col(2,blue) <- true.

win :- heads, col(_,red).
win :- col(1,C), col(2,C).
```

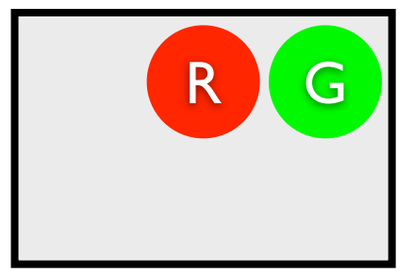
$0.4 \times 0.3 \times 0.3$



$(1-0.4) \times 0.3 \times 0.2$



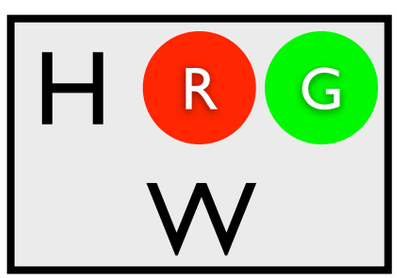
$(1-0.4) \times 0.3 \times 0.3$



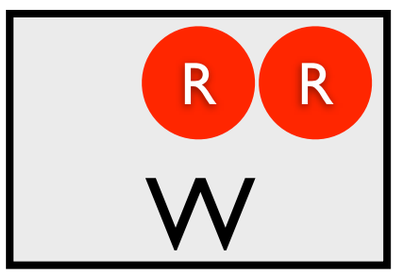
Possible Worlds

```
0.4 :: heads.

0.3 :: col(1,red); 0.7 :: col(1,blue) <- true.
0.2 :: col(2,red); 0.3 :: col(2,green); 0.5 :: col(2,blue) <- true.

win :- heads, col(_,red).
win :- col(1,C), col(2,C).
```

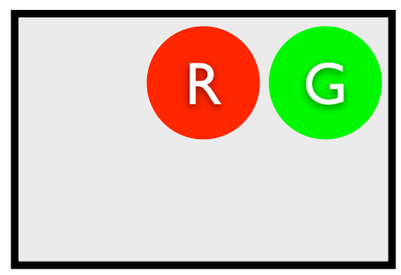
0.4 ×0.3 ×0.3



(1−0.4)×0.3 ×0.2



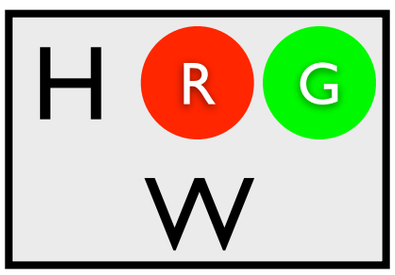
(1−0.4)×0.3 ×0.3



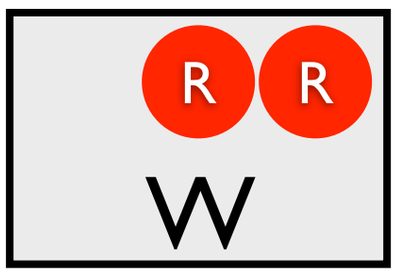
Possible Worlds

```
0.4 :: heads.

0.3 :: col(1,red); 0.7 :: col(1,blue) <- true.
0.2 :: col(2,red); 0.3 :: col(2,green); 0.5 :: col(2,blue) <- true.

win :- heads, col( ,red).
win :- col(1,C), col(2,C).
```

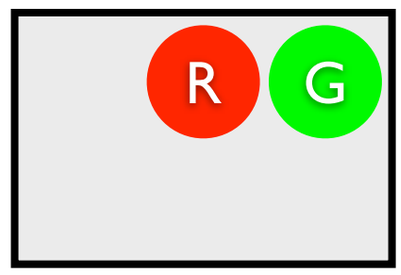
0.4 ×0.3 ×0.3



(1−0.4)×0.3 ×0.2

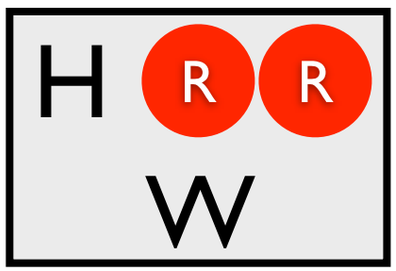


(1−0.4)×0.3 ×0.3

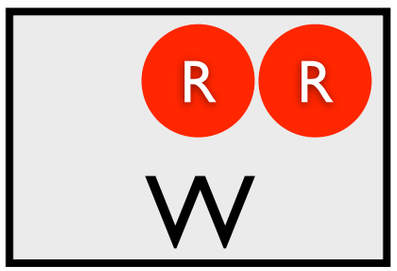


All Possible Worlds

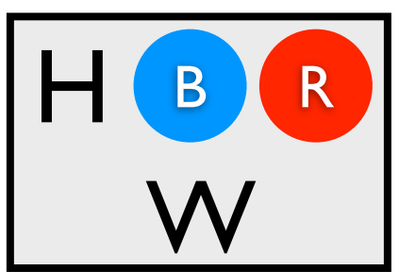
0.024



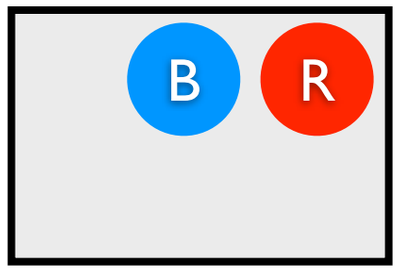
0.036



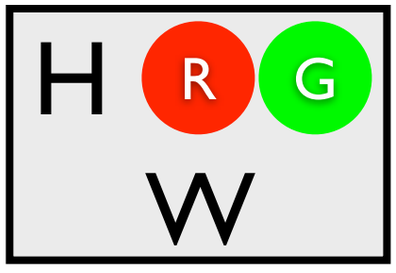
0.056



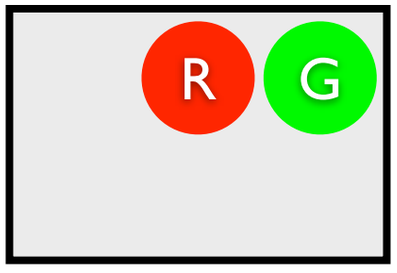
0.084



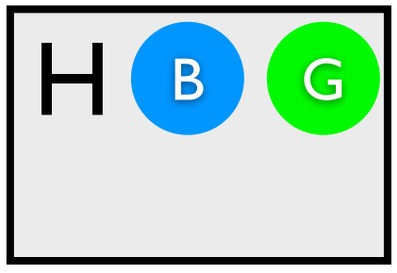
0.036



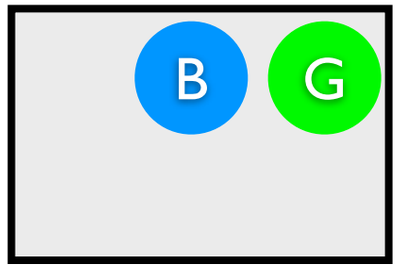
0.054



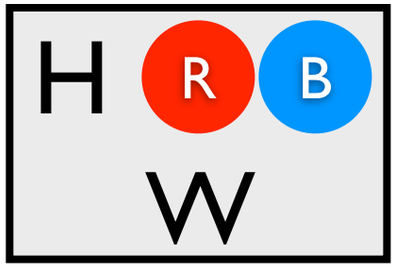
0.084



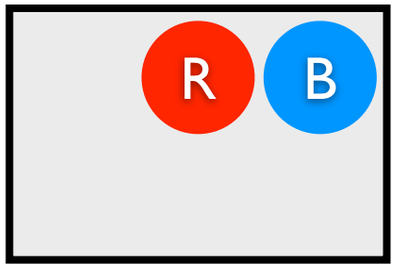
0.126



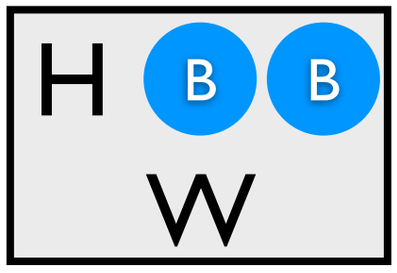
0.060



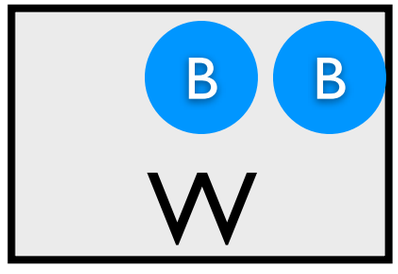
0.090



0.140

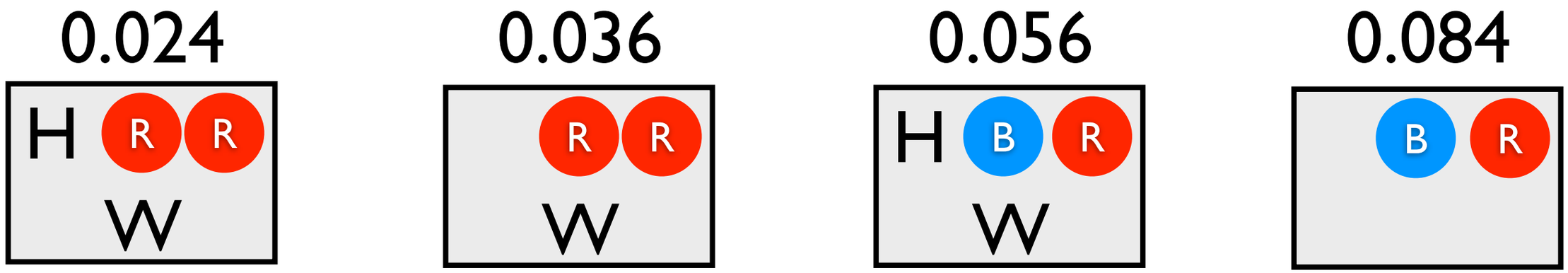


0.210



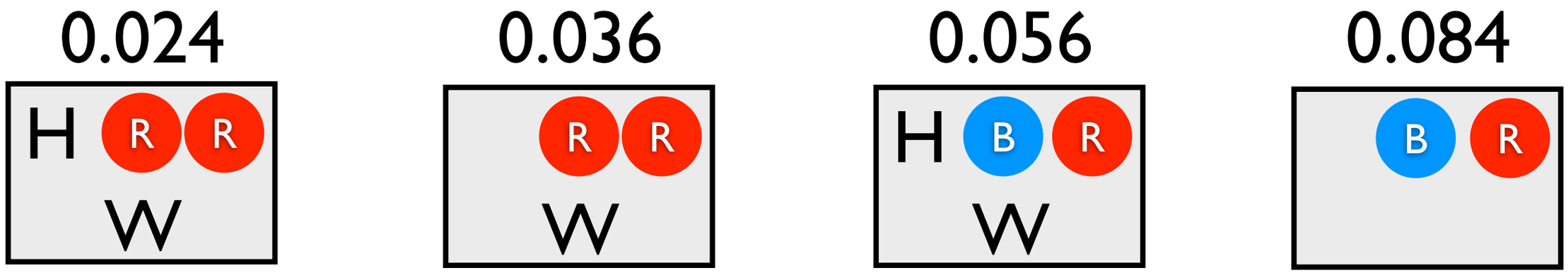
Most likely world where **win** is true?

MPE Inference



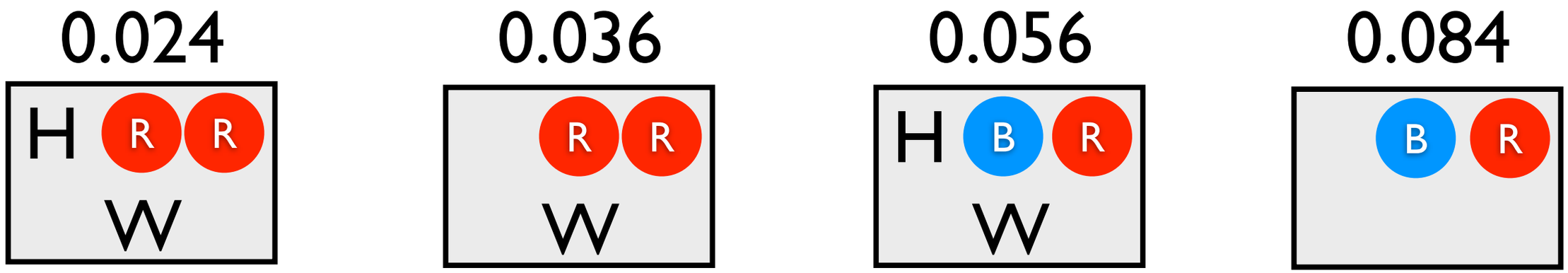
Most likely world where **win** is true?

MPE Inference



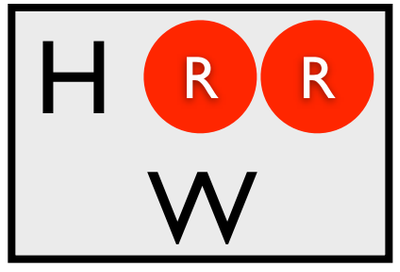
Most likely world where `col(2,blue)` is false?

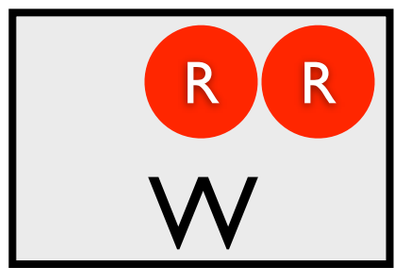
MPE Inference

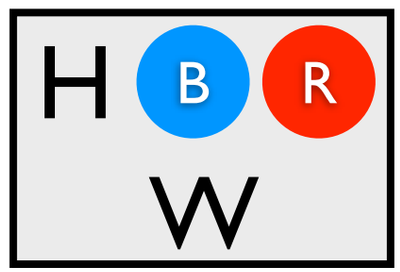


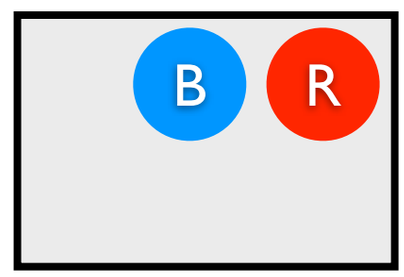
Most likely world where `col(2,blue)` is false?

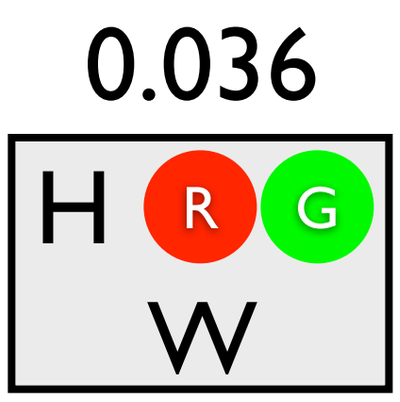
MPE Inference

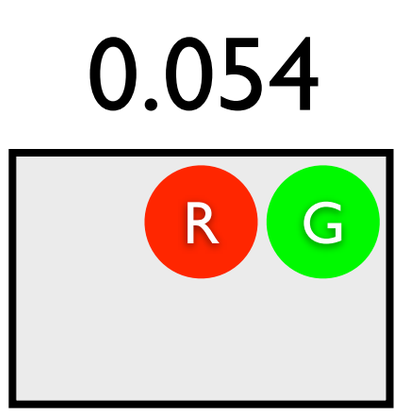


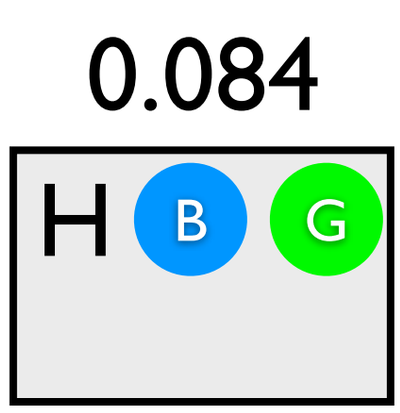


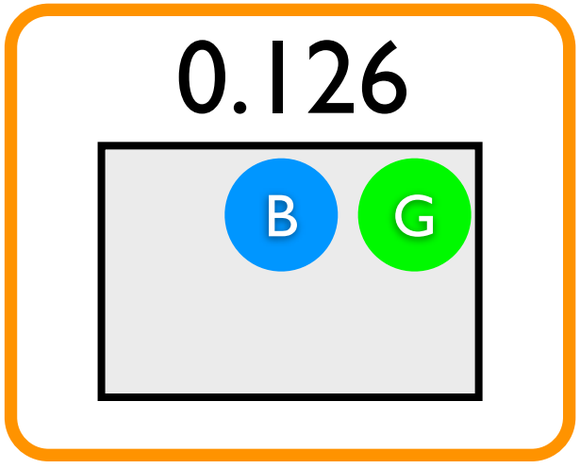


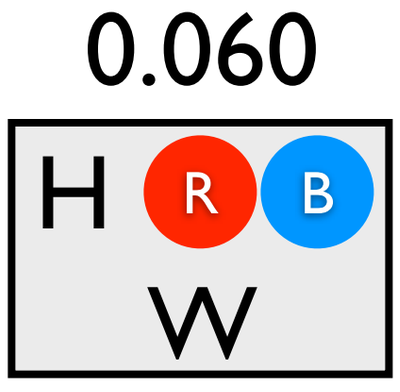


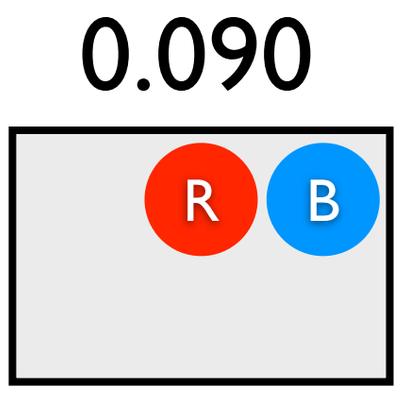


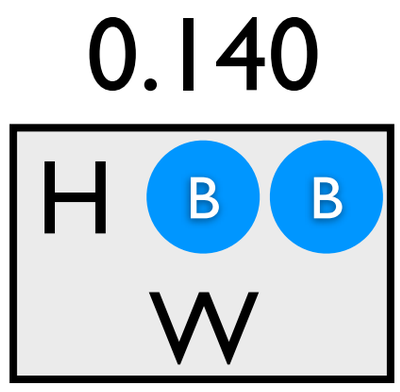


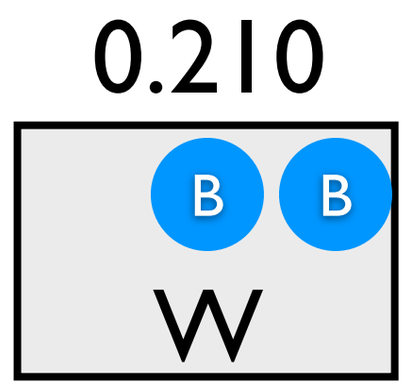






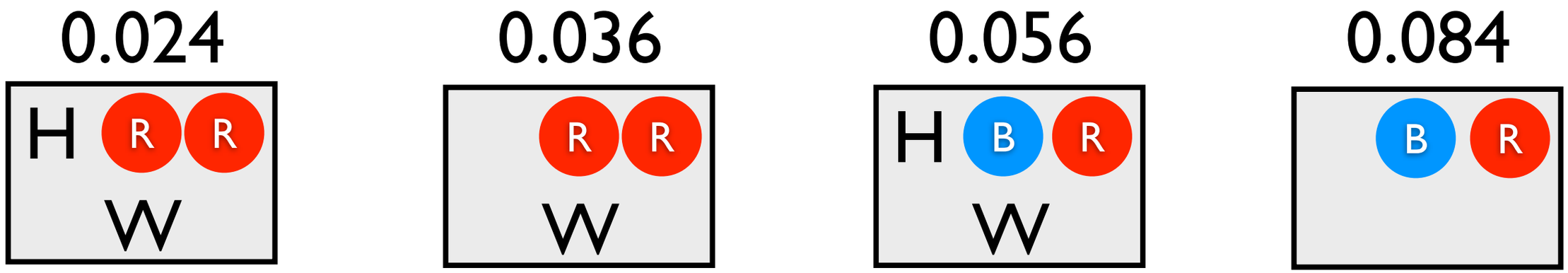






P(win)=?

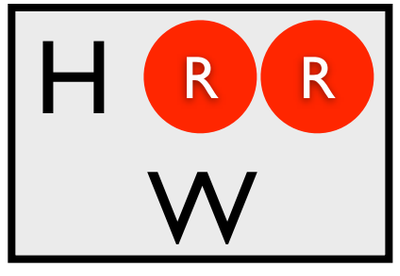
Marginal Probability



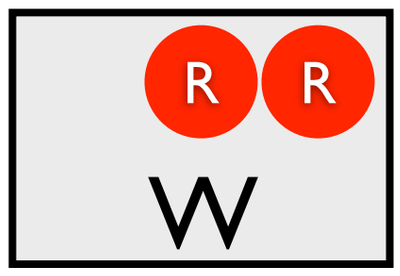
P(win)= Σ

Marginal Probability

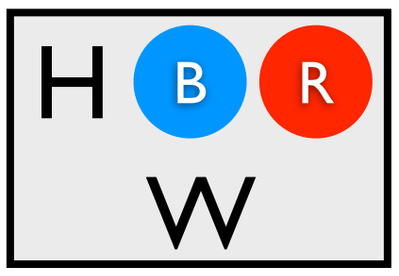
0.024



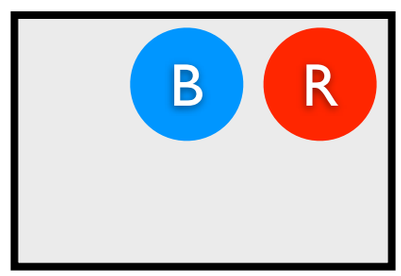
0.036



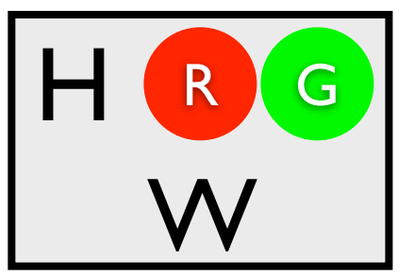
0.056



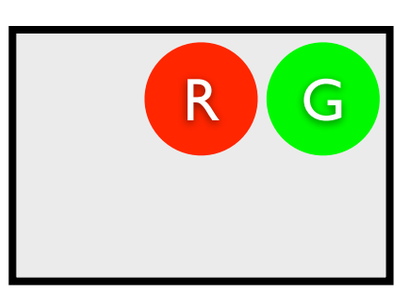
0.084



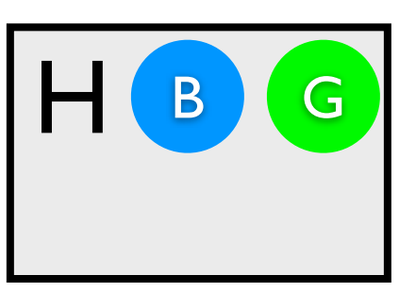
0.036



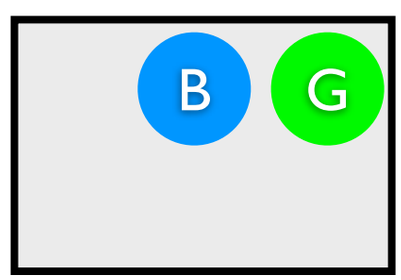
0.054



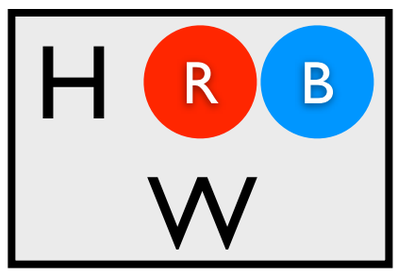
0.084



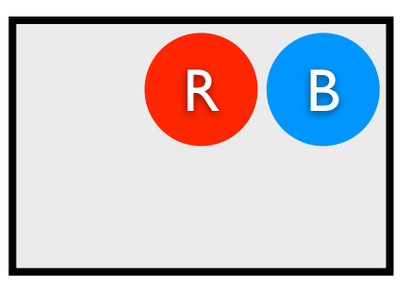
0.126



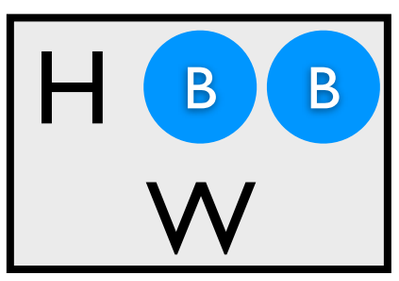
0.060



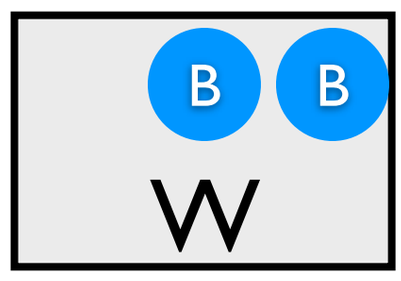
0.090



0.140



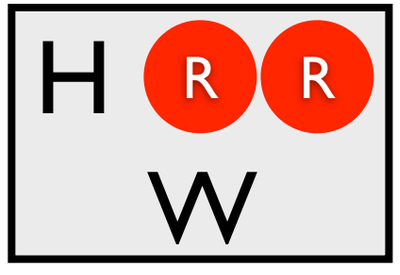
0.210



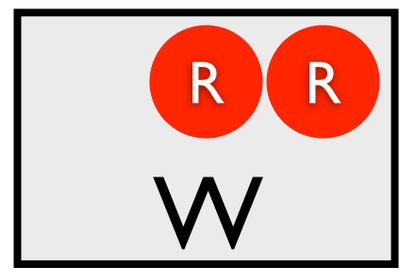
P(win)= Σ =0.562

Marginal Probability

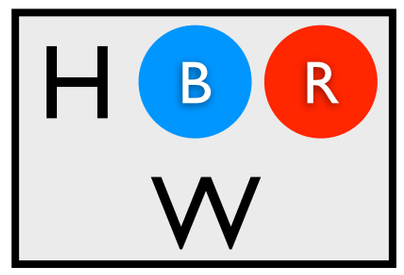
0.024



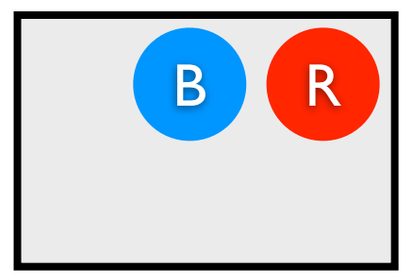
0.036



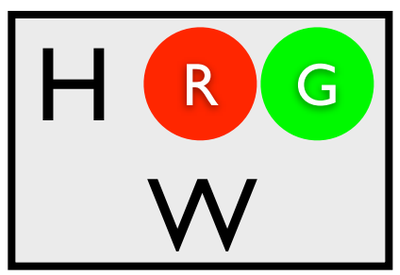
0.056



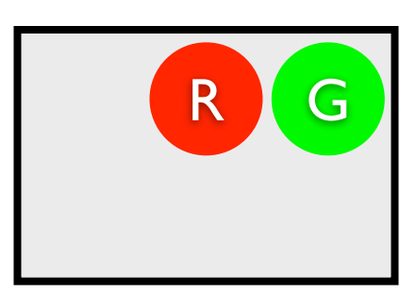
0.084



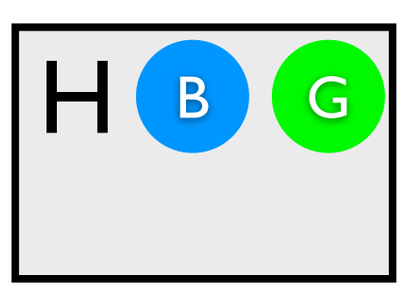
0.036



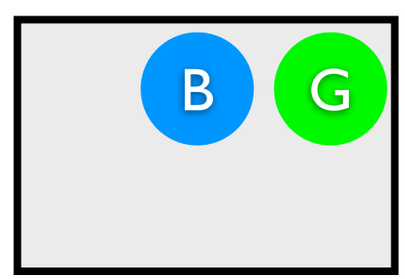
0.054



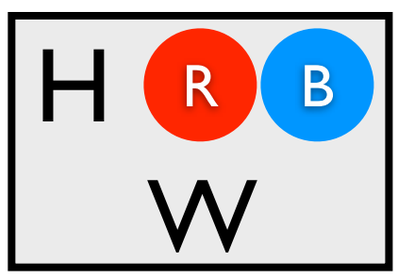
0.084



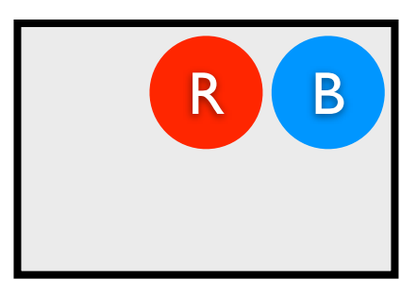
0.126



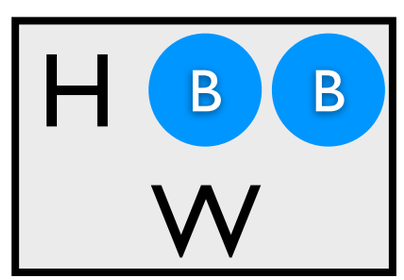
0.060



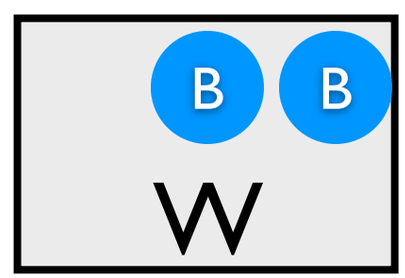
0.090



0.140



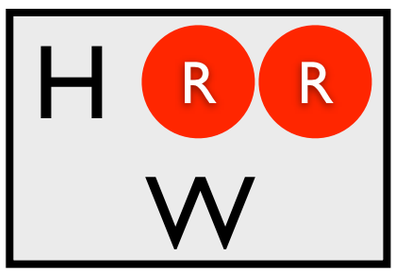
0.210



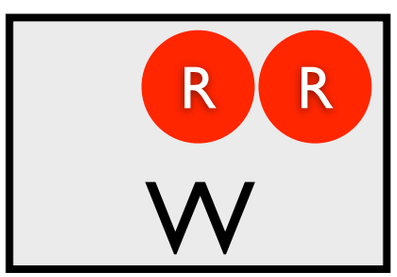
P(win|col(2,green))= ?

Conditional Probability

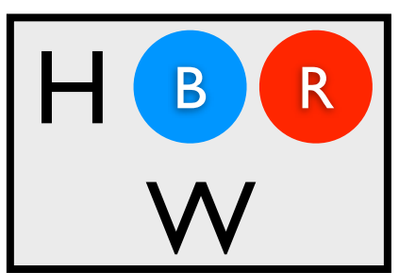
0.024



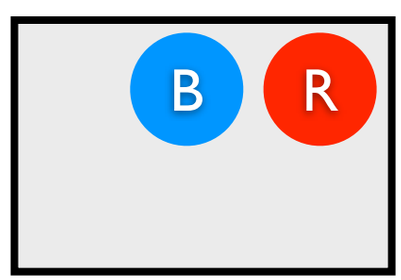
0.036



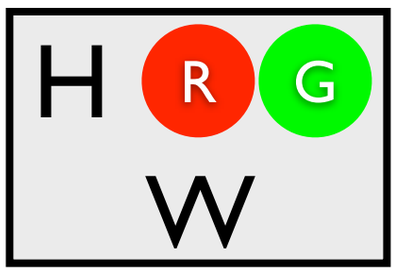
0.056



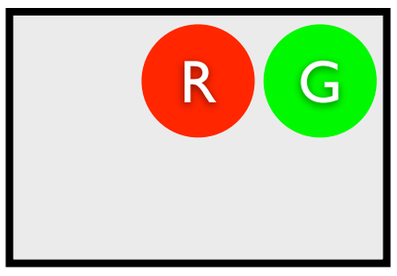
0.084



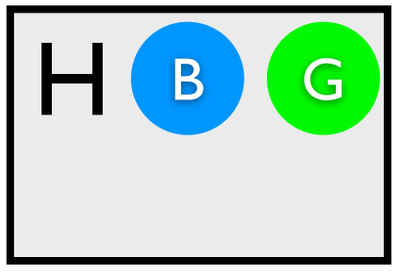
0.036



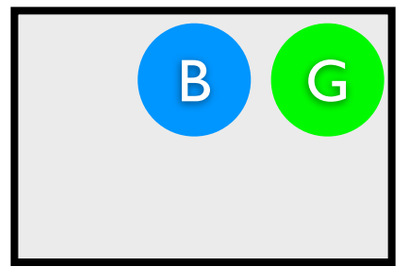
0.054



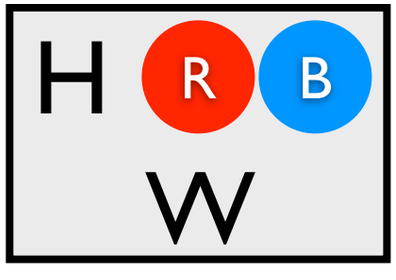
0.084



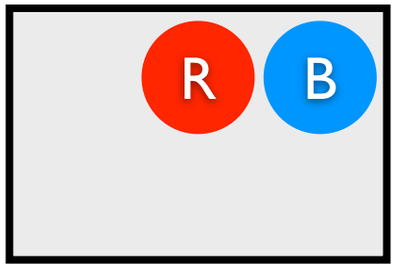
0.126



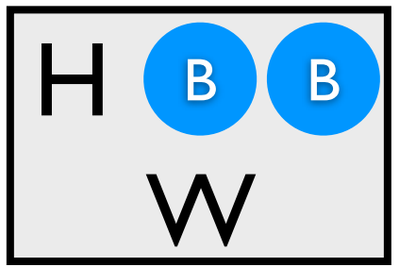
0.060



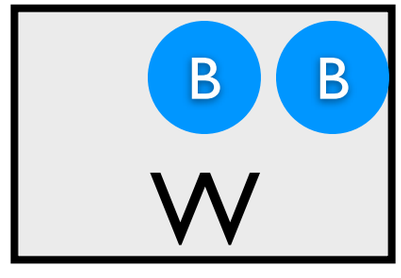
0.090



0.140



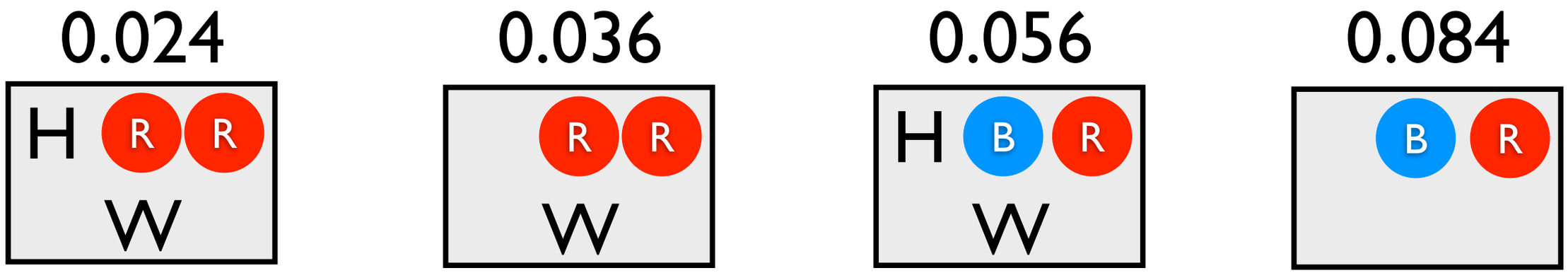
0.210



$$P(win|col(2,green)) = \Sigma/\Sigma$$

$$= P(win \wedge col(2,green))/P(col(2,green))$$

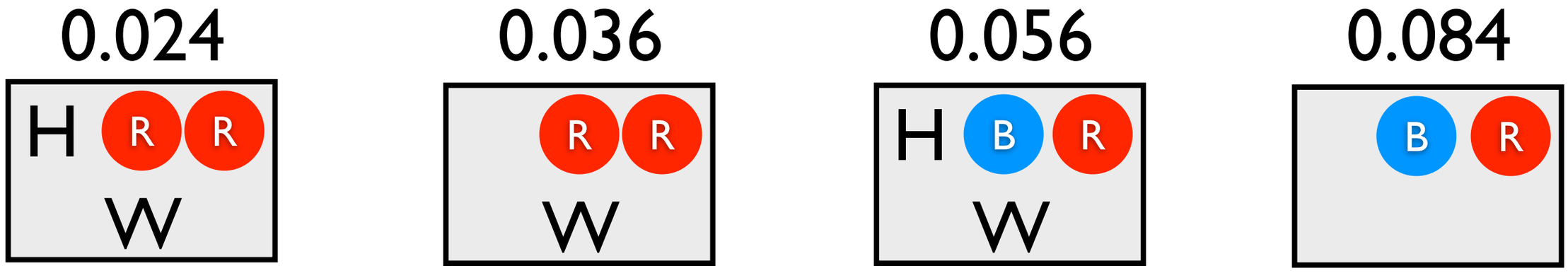
Conditional Probability



$$P(win|col(2,green)) = \Sigma/\Sigma$$

$$= P(win \wedge col(2,green))/P(col(2,green))$$

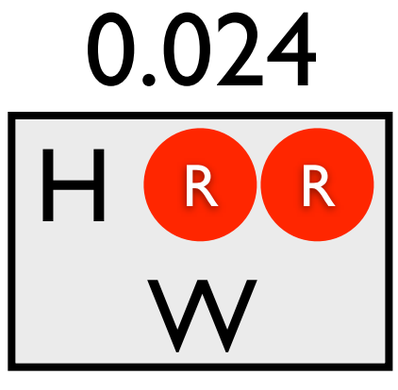
Conditional Probability

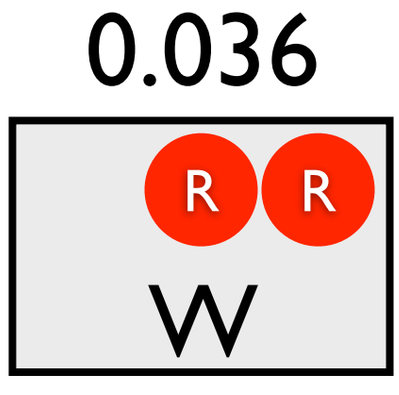


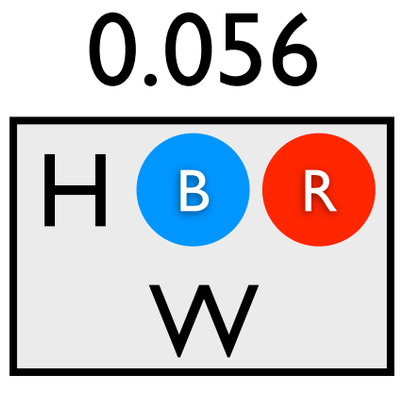
P(win|col(2,green))= ∑/∑

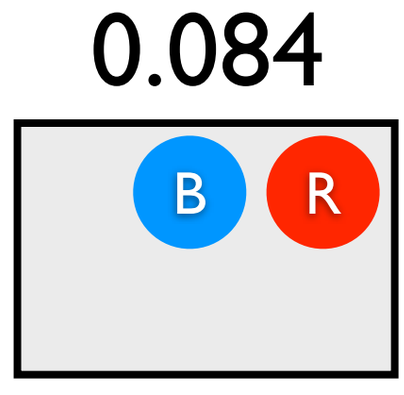
=0.036/0.3=0.12

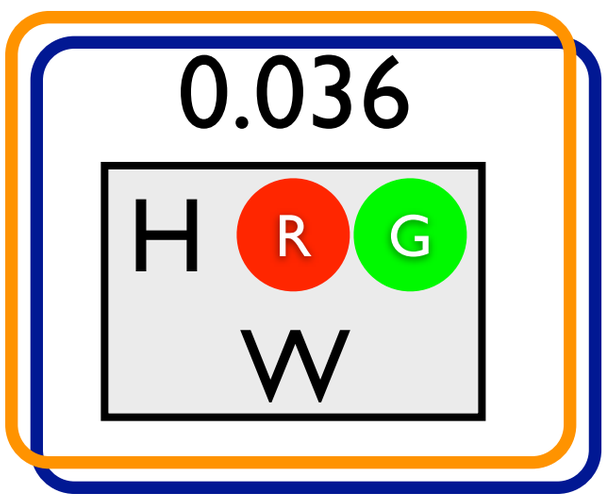
Conditional Probability

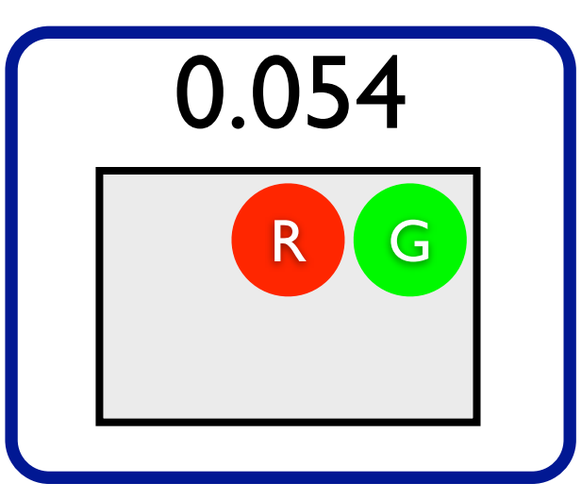


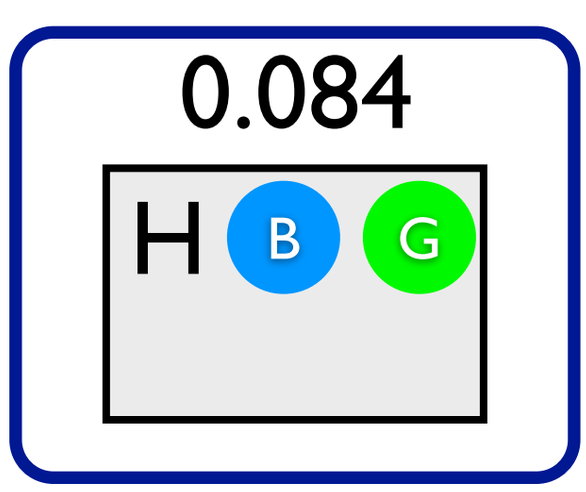


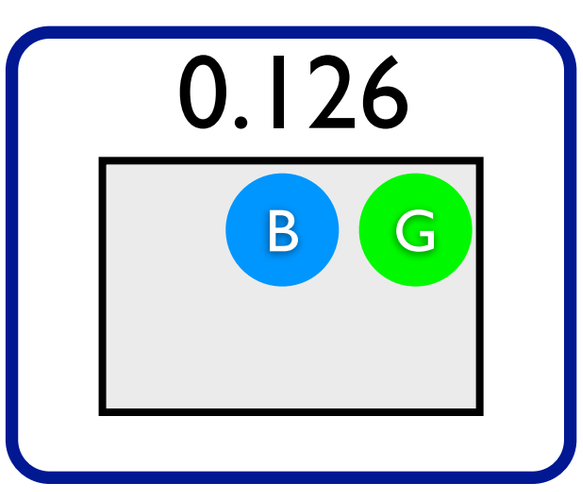


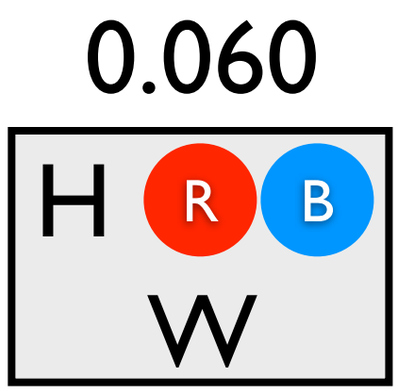


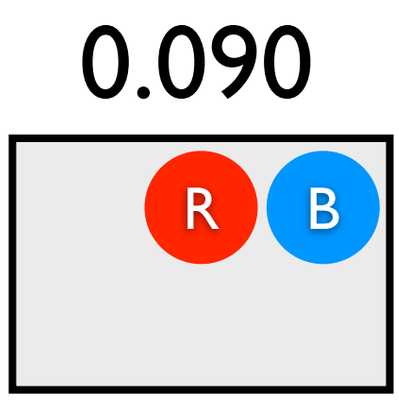


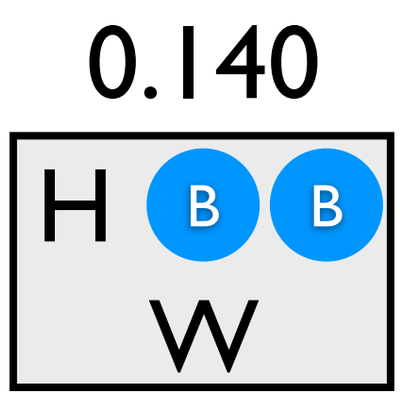


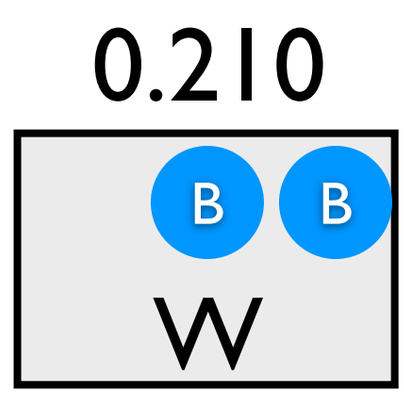












Distribution Semantics

(with probabilistic facts)

[Sato, ICLP 95]

$$P(Q) = \sum_{F \cup R \models Q} \prod_{f \in F} p(f) \prod_{f \notin F} 1 - p(f)$$

Distribution Semantics

(with probabilistic facts)

[Sato, ICLP 95]

query

$$P(Q) = \sum_{F \cup R \models Q} \prod_{f \in F} p(f) \prod_{f \notin F} 1 - p(f)$$

Distribution Semantics

(with probabilistic facts)

[Sato, ICLP 95]

query

$$P(Q) = \sum_{F \cup R \models Q} \prod_{f \in F} p(f) \prod_{f \notin F} 1 - p(f)$$

subset of probabilistic facts

Distribution Semantics

(with probabilistic facts)

[Sato, ICLP 95]

query

$$P(Q) = \sum_{F \cup R \models Q} \prod_{f \in F} p(f) \prod_{f \notin F} 1 - p(f)$$

subset of probabilistic facts

Prolog rules

Distribution Semantics

(with probabilistic facts)

[Sato, ICLP 95]

query

sum over possible worlds where Q is true

$$P(Q) = \sum_{F \cup R \models Q} \prod_{f \in F} p(f) \prod_{f \notin F} 1 - p(f)$$

subset of probabilistic facts

Prolog rules

Distribution Semantics

(with probabilistic facts)

[Sato, ICLP 95]

query

sum over possible worlds where Q is true

$$P(Q) = \sum_{F \cup R \models Q} \prod_{f \in F} p(f) \prod_{f \notin F} 1 - p(f)$$

subset of probabilistic facts

Prolog rules

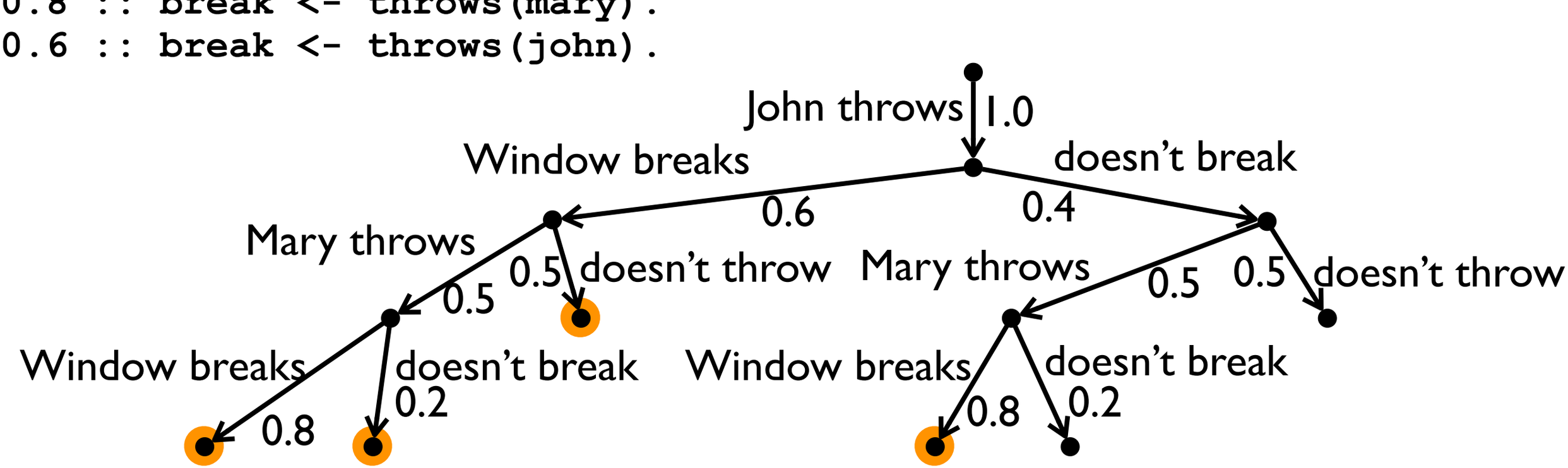
probability of possible world

[Vennekens et al, ICLP 04]

Alternative view: CP-Logic

```
throws(john).
0.5::throws(mary).

0.8 :: break <- throws(mary).
0.6 :: break <- throws(john).
```

probabilistic causal laws



P(break)=0.6×0.5×0.8+0.6×0.5×0.2+0.6×0.5+0.4×0.5×0.8

Answering Questions

Given:

program

queries

evidence

?

Find:

marginal probabilities

conditional probabilities

MPE state

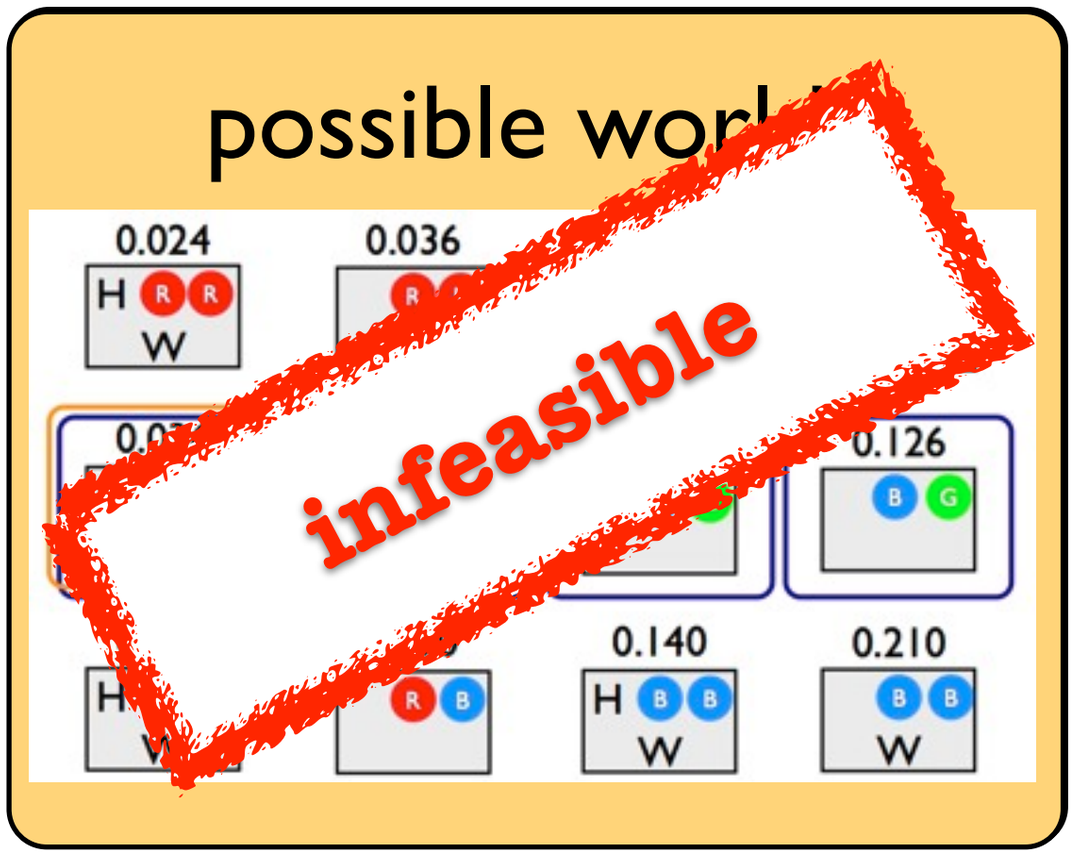
Answering Questions

Given:

program

queries

evidence



Find:

marginal probabilities

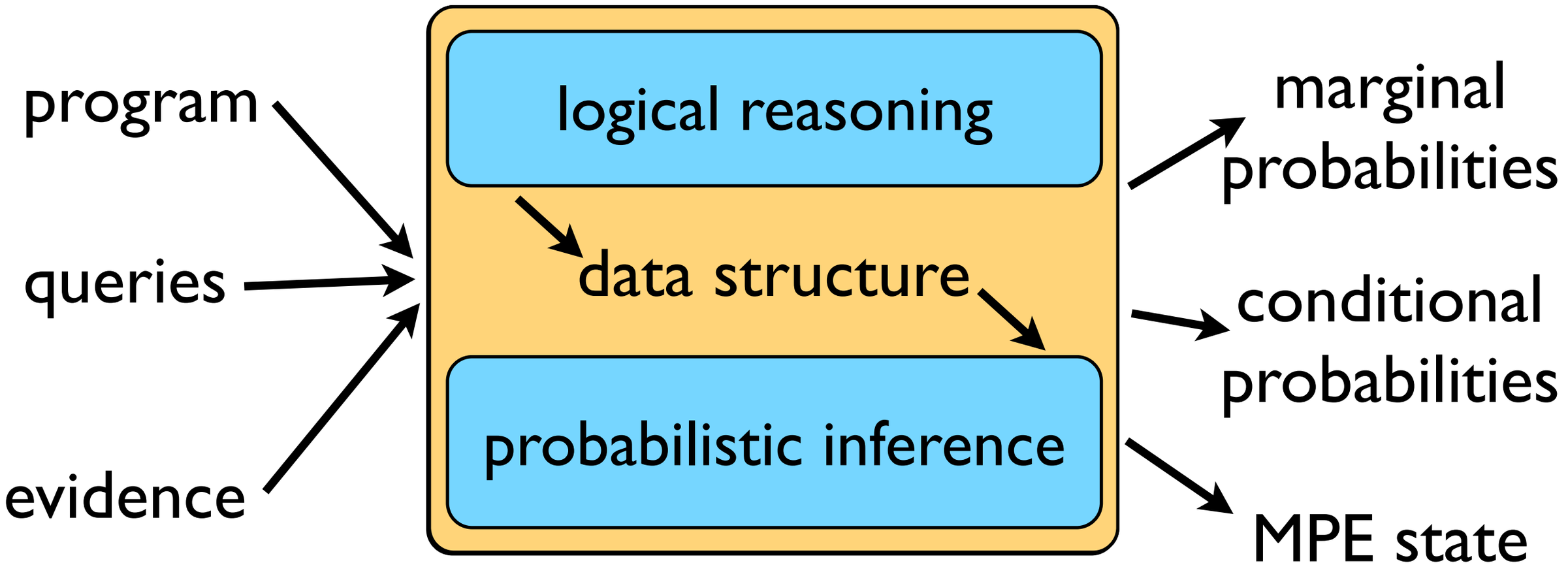
conditional probabilities

MPE state

Answering Questions

Given:

Find:

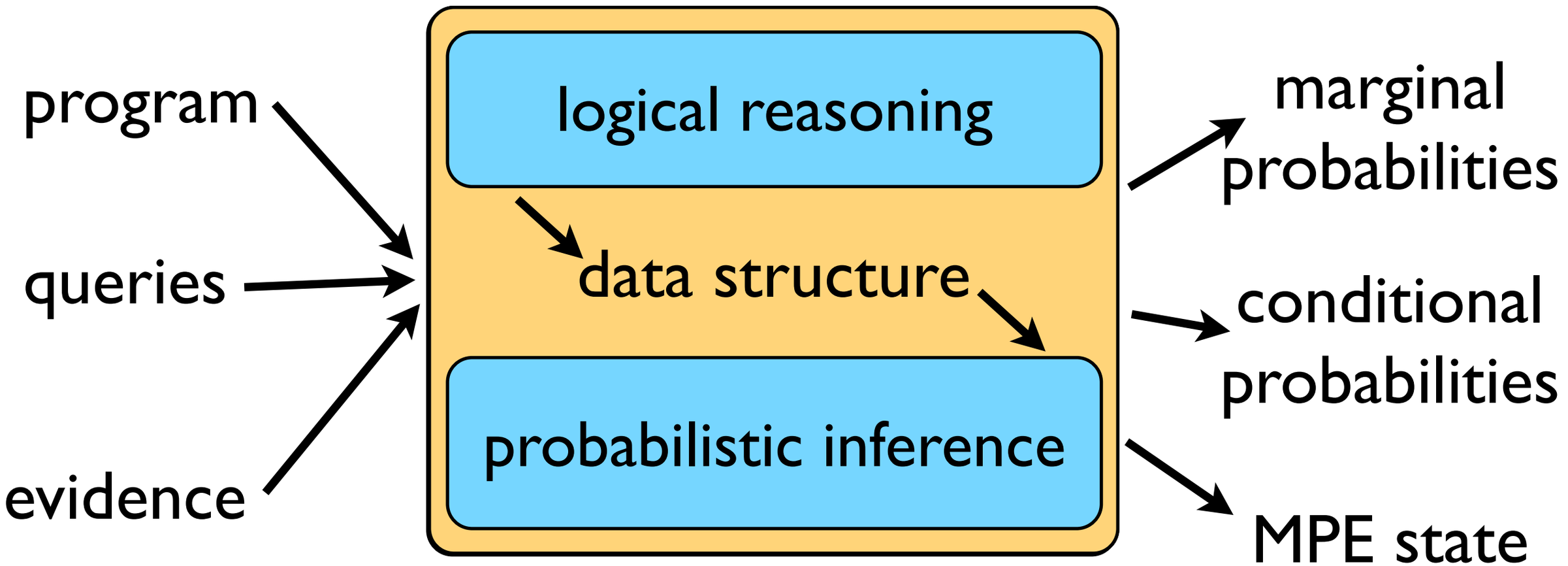


Answering Questions

more on this later

Given:

Find:



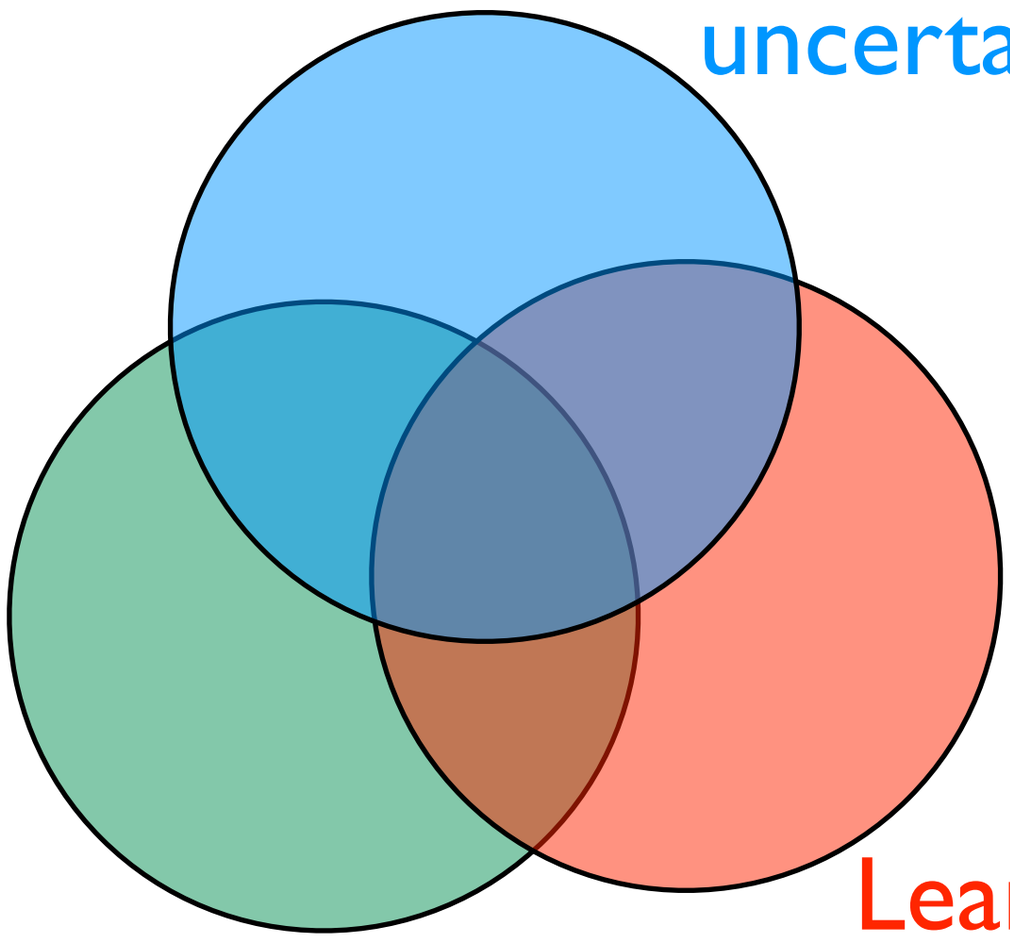
[Suciu et al 2011]

Probabilistic Databases

Dealing with uncertainty

Reasoning with relational data

Learning



[Suciu et al 2011]

Probabilistic Databases

```
select x.person, y.country
from bornIn x, cityIn y
where x.city=y.city
```

bornIn

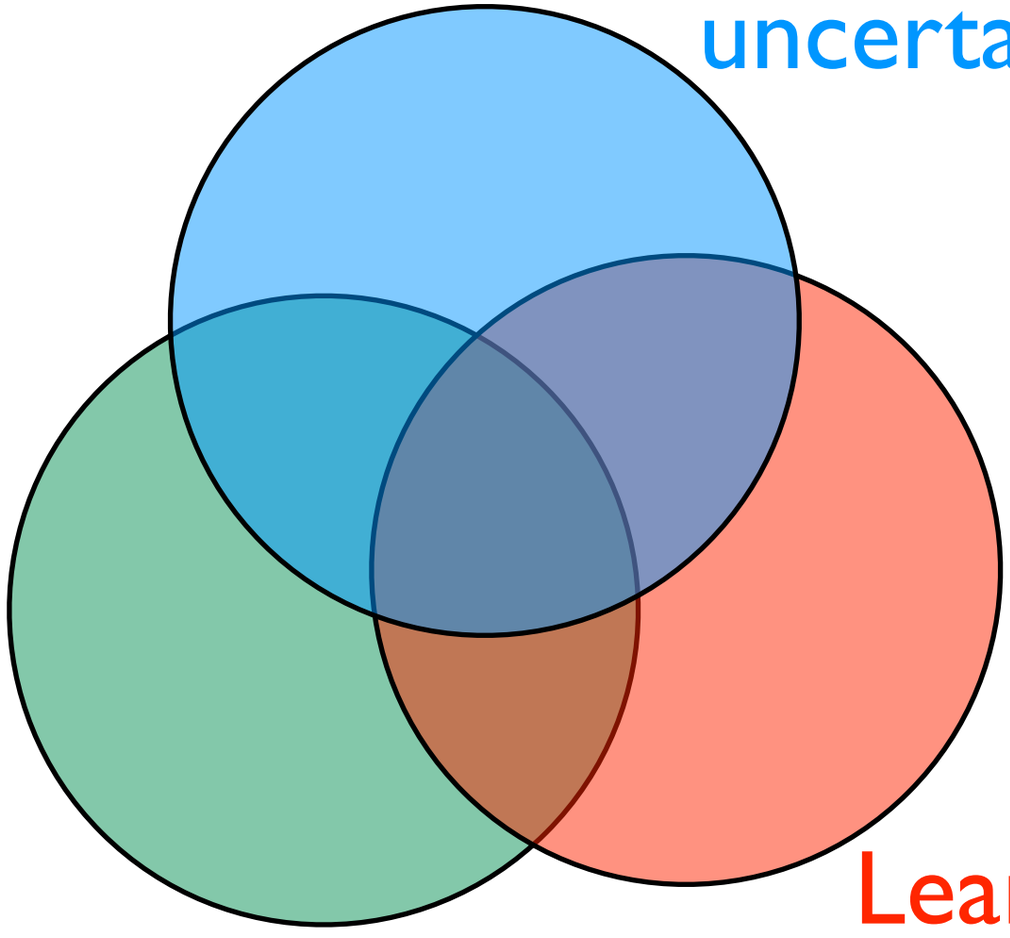
person	city
ann	london
bob	york
eve	new york
tom	paris

cityIn

city	country
london	uk
york	uk
paris	usa

relational database

Dealing with uncertainty



Learning

[Suciu et al 2011]

Probabilistic Databases

```
select x.person, y.country
from bornIn x, cityIn y
where x.city=y.city
```

one world

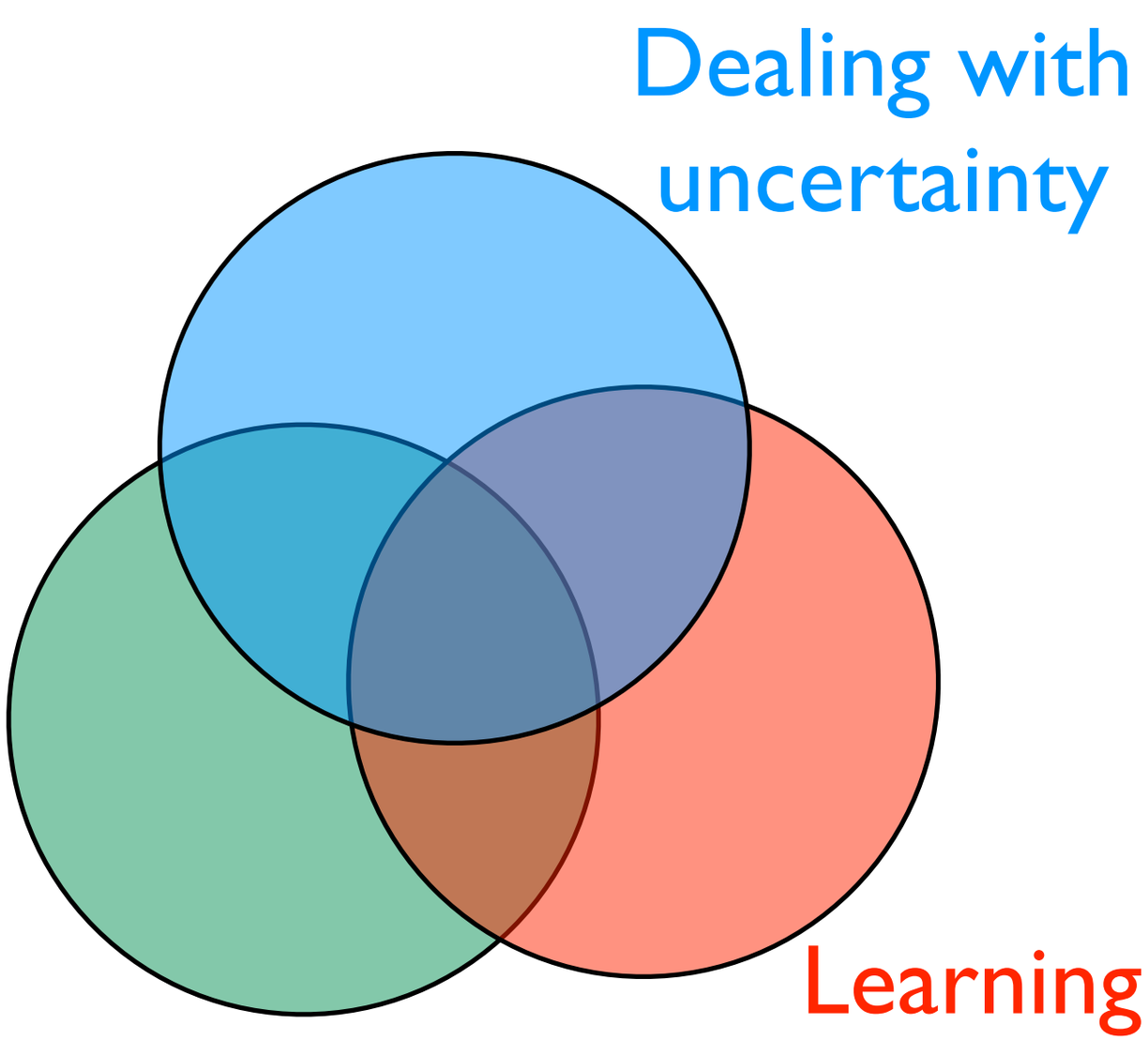
bornIn

person	city
ann	london
bob	york
eve	new york
tom	paris

cityIn

city	country
london	uk
york	uk
paris	usa

relational database



[Suciu et al 2011]

Probabilistic Databases

bornIn

person	city	P
ann	london	0.87
bob	york	0.95
eve	new york	0.90
tom	paris	0.56

cityIn

city	country	P
london	uk	0.99
york	uk	0.75
paris	usa	0.40

tuples as random variables

```
select x.person, y.country
from bornIn x, cityIn y
where x.city=y.city
```

one world

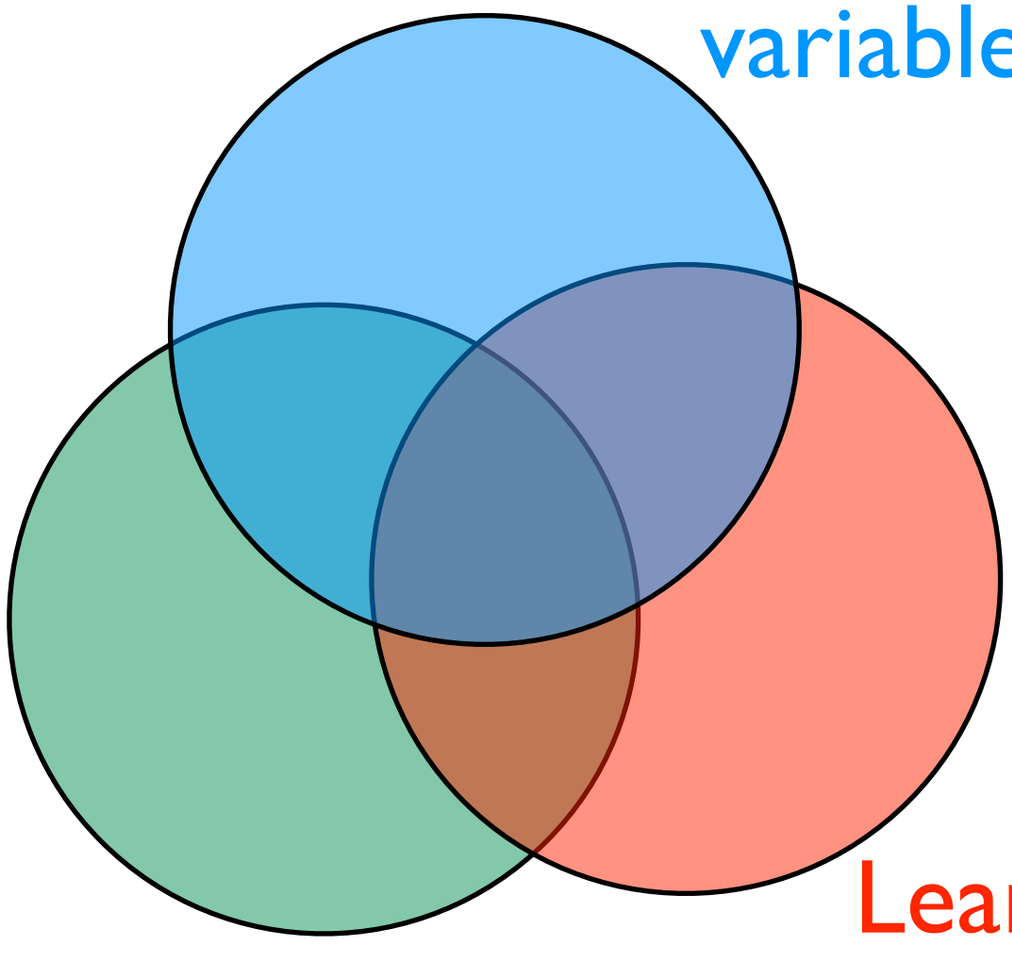
bornIn

person	city
ann	london
bob	york
eve	new york
tom	paris

cityIn

city	country
london	uk
york	uk
paris	usa

relational database



Learning

[Suciu et al 2011]

Probabilistic Databases

several possible worlds

bornIn

person	city	P
ann	london	0.87
bob	york	0.95
eve	new york	0.90
tom	paris	0.56

cityIn

city	country	P
london	uk	0.99
york	uk	0.75
paris	usa	0.40

tuples as random

```
select x.person, y.country
from bornIn x, cityIn y
where x.city=y.city
```

one world

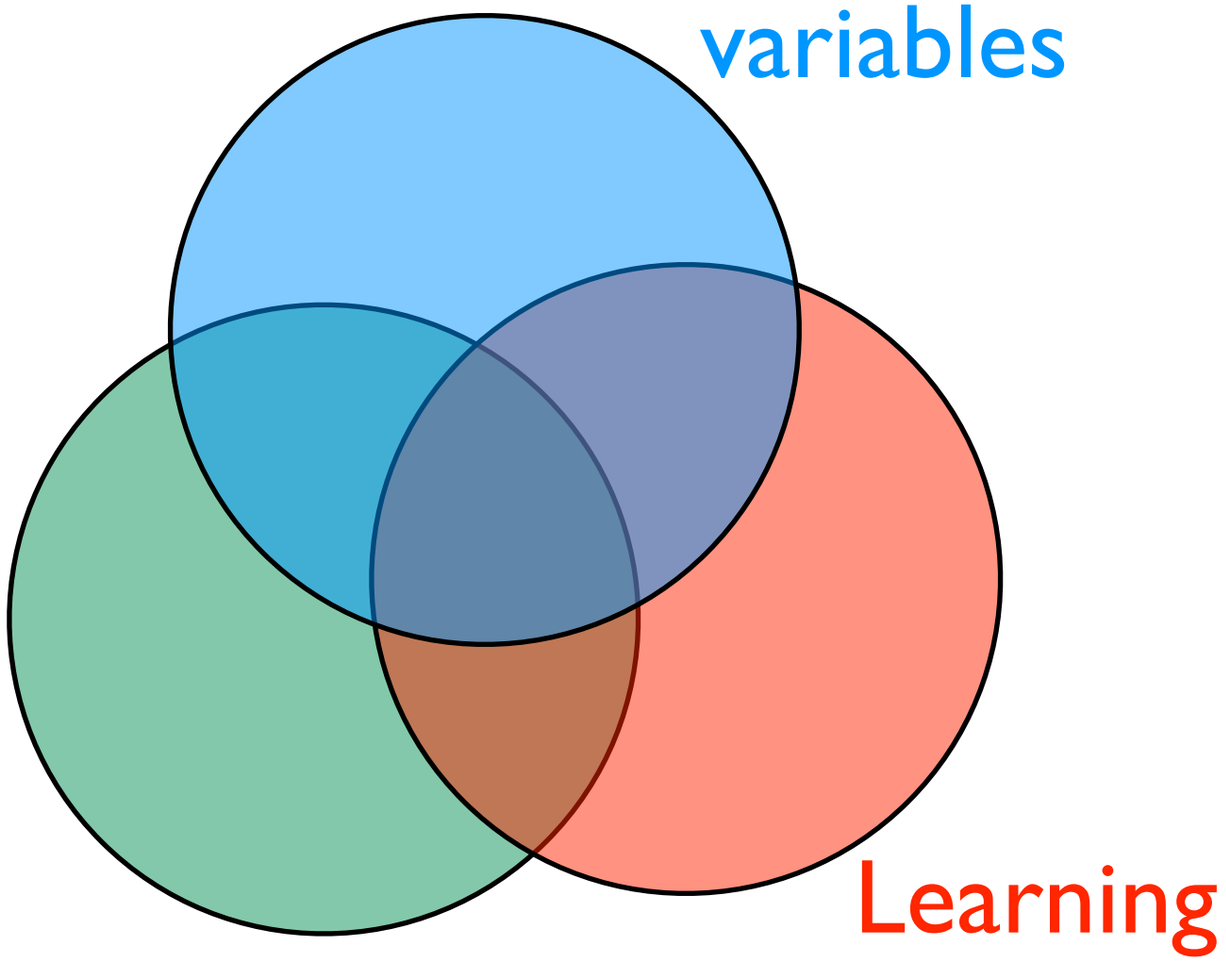
bornIn

person	city
ann	london
bob	york
eve	new york
tom	paris

cityIn

city	country
london	uk
york	uk
paris	usa

relational database



[Suciu et al 2011]

Probabilistic Databases

several possible worlds

bornIn

person	city	P
ann	london	0.87
bob	york	0.95
		0.90
		0.56

cityIn

city	country	P
london	uk	0.99
york	uk	0.75
paris	usa	0.40

tuples as random

probabilistic tables + database queries
→ distribution over possible worlds

```
select
from bornIn x, cityIn y
where x.city=y.city
```

one world

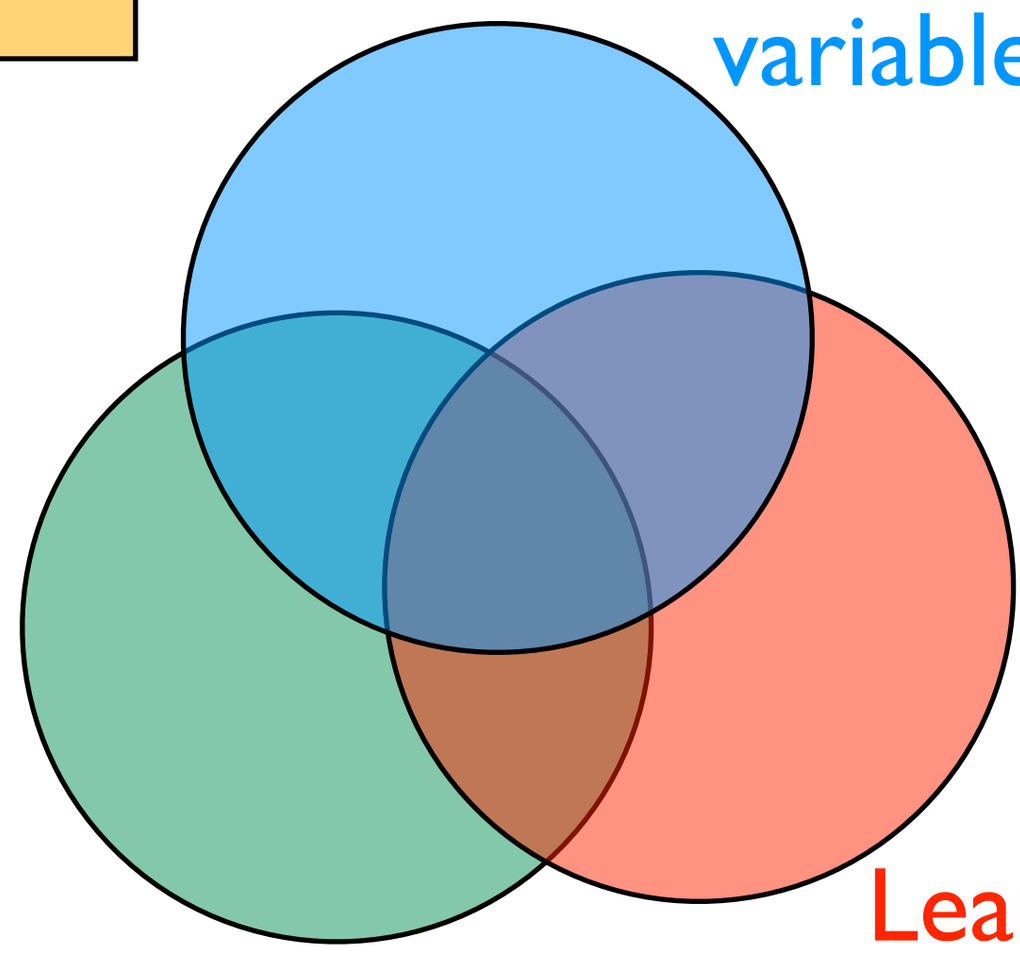
bornIn

person	city
ann	london
bob	york
eve	new york
tom	paris

cityIn

city	country
london	uk
york	uk
paris	usa

relational database



Example: Information Extraction

Recently-Learned Facts twitter Refresh

instance	iteration	date learned	confidence
kelly_andrews is a female	826	29-mar-2014	98.7
investment_next_year is an economic sector	829	10-apr-2014	95.3
shibenik is a geopolitical entity that is an organization	829	10-apr-2014	97.2
quality_web_design_work is a character trait	826	29-mar-2014	91.0
mercedes_benz_cls_by_carlsson is an automobile manufacturer	829	10-apr-2014	95.2
social_work is an academic program at the university rutgers_university	827	02-apr-2014	93.8
dante wrote the book the_divine_comedy	826	29-mar-2014	93.8
willie_aames was born in the city los_angeles	831	16-apr-2014	100.0
kitt_peak is a mountain in the state or province arizona	831	16-apr-2014	96.9
greenwich is a park in the city london	831	16-apr-2014	100.0

instances for many different relations

degree of certainty

NELL: http://rtw.ml.cmu.edu/rtw/

Querying: relational database

ProducesProduct

Company	Product
sony	walkman
microsoft	mac_os_x
ibm	personal_computer
microsoft	mac_os
adobe	adobe_indesign
adobe	adobe_dreamweaver
...	...

HeadquarteredIn

Company	City
microsoft	redmond
ibm	san_jose
emirates_airlines	dubai
honda	torrance
horizon	seattle
egyptair	cairo
adobe	san_jose
...	...

[Example from Suciu et al 2011]

Querying: relational database

ProducesProduct

Company	Product
sony	walkman
microsoft	mac_os_x
ibm	personal_computer
microsoft	mac_os
adobe	adobe_indesign
adobe	adobe_dreamweaver
...	...

HeadquarteredIn

Company	City
microsoft	redmond
ibm	san_jose
emirates_airlines	dubai
honda	torrance
horizon	seattle
egyptair	cairo
adobe	san_jose
...	...

```
select x.Product, x.Company
from ProducesProduct x, HeadquarteredIn y
where x.Company=y.Company and
y.City='san_jose'
```


[Example from Suciu et al 2011]

Querying: relational database

ProducesProduct

Company	Product
sony	walkman
microsoft	mac_os_x
ibm	personal_computer
microsoft	mac_os
adobe	adobe_indesign
adobe	adobe_dreamweaver
...	...

HeadquarteredIn

Company	City
microsoft	redmond
ibm	san_jose
emirates_airlines	dubai
honda	torrance
horizon	seattle
egyptair	cairo
adobe	san_jose
...	...

```
select x.Product, x.Company
from ProducesProduct x, HeadquarteredIn y
where x.Company=y.Company and
y.City='san_jose'
```

Product	Company
personal_computer	ibm
adobe_indesign	adobe
adobe_dreamweaver	adobe

[Example from Suciu et al 2011]

Querying: relational database

ProducesProduct

Company	Product
sony	walkman
microsoft	mac_os_x
ibm	personal_computer
microsoft	mac_os
adobe	adobe_indesign
adobe	adobe_dreamweaver
...	...

HeadquarteredIn

Company	City
microsoft	redmond
ibm	san_jose
emirates_airlines	dubai
honda	torrance
horizon	seattle
egyptair	cairo
adobe	san_jose
...	...

```
select x.Product, x.Company
from ProducesProduct x, HeadquarteredIn y
where x.Company=y.Company and
y.City='san_jose'
```

Product	Company
personal_computer	ibm
adobe_indesign	adobe
adobe_dreamweaver	adobe

[Example from Suciu et al 2011]

Querying: relational database

ProducesProduct

Company	Product
sony	walkman
microsoft	mac_os_x
ibm	personal_computer
microsoft	mac_os
adobe	adobe_indesign
adobe	adobe_dreamweaver
...	...

HeadquarteredIn

Company	City
microsoft	redmond
ibm	san_jose
emirates_airlines	dubai
honda	torrance
horizon	seattle
egyptair	cairo
adobe	san_jose
...	...

```
select x.Product, x.Company
from ProducesProduct x, HeadquarteredIn y
where x.Company=y.Company and
y.City='san_jose'
```

Product	Company
personal_computer	ibm
adobe_indesign	adobe
adobe_dreamweaver	adobe

[Example from Suciu et al 2011]

Querying: relational database

ProducesProduct

Company	Product
sony	walkman
microsoft	mac_os_x
ibm	personal_computer
microsoft	mac_os
adobe	adobe_indesign
adobe	adobe_dreamweaver
...	...

HeadquarteredIn

Company	City
microsoft	redmond
ibm	san_jose
emirates_airlines	dubai
honda	torrance
horizon	seattle
egyptair	cairo
adobe	san_jose
...	...

```
select x.Product, x.Company
from ProducesProduct x, HeadquarteredIn y
where x.Company=y.Company and
y.City='san_jose'
```

Product	Company
personal_computer	ibm
adobe_indesign	adobe
adobe_dreamweaver	adobe

[Example from Suciu et al 2011]

Querying: probabilistic db

ProducesProduct

Company	Product	P
sony	walkman	0.96
microsoft	mac_os_x	0.96
ibm	personal_computer	0.96
microsoft	mac_os	0.9
adobe	adobe_indesign	0.9
adobe	adobe_dreamweaver	0.87
...	...	...

HeadquarteredIn

Company	City	P
microsoft	redmond	1.00
ibm	san_jose	0.99
emirates_airlines	dubai	0.93
honda	torrance	0.93
horizon	seattle	0.93
egyptair	cairo	0.93
adobe	san_jose	0.93
...	...	...

[Example from Suciu et al 2011]

Querying: probabilistic db

ProducesProduct

Company	Product	P
sony	walkman	0.96
microsoft	mac_os_x	0.96
ibm	personal_computer	0.96
microsoft	mac_os	0.9
adobe	adobe_indesign	0.9
adobe	adobe_dreamweaver	0.87
...	...	...

HeadquarteredIn

Company	City	P
microsoft	redmond	1.00
ibm	san_jose	0.99
emirates_airlines	dubai	0.93
honda	torrance	0.93
horizon	seattle	0.93
egyptair	cairo	0.93
adobe	san_jose	0.93
...	...	...

```
select x.Product, x.Company
from ProducesProduct x, HeadquarteredIn y
where x.Company=y.Company and
y.City='san_jose'
```

same query -

probabilities handled implicitly

Product	Company	P
personal_computer	ibm	0.95
adobe_indesign	adobe	0.83
adobe_dreamweaver	adobe	0.80

[Example from Suciu et al 2011]

Querying: probabilistic db

ProducesProduct

Company	Product	P
sony	walkman	0.96
microsoft	mac_os_x	0.96
ibm	personal_computer	0.96
microsoft	mac_os	0.9
adobe	adobe_indesign	0.9
adobe	adobe_dreamweaver	0.87
...	...	...

HeadquarteredIn

Company	City	P
microsoft	redmond	1.00
ibm	san_jose	0.99
emirates_airlines	dubai	0.93
honda	torrance	0.93
horizon	seattle	0.93
egyptair	cairo	0.93
adobe	san_jose	0.93
...	...	...

```
select x.Product, x.Company
from ProducesProduct x, HeadquarteredIn y
where x.Company=y.Company and
y.City='san_jose'
```

0.96x0.99=0.95

Product	Company	P
personal_computer	ibm	0.95
adobe_indesign	adobe	0.83
adobe_dreamweaver	adobe	0.80

[Example from Suciu et al 2011]

Querying: probabilistic db

ProducesProduct

Company	Product	P
sony	walkman	0.96
microsoft	mac_os_x	0.96
ibm	personal_computer	0.96
microsoft	mac_os	0.9
adobe	adobe_indesign	0.9
adobe	adobe_dreamweaver	0.87
...	...	...

HeadquarteredIn

Company	City	P
microsoft	redmond	1.00
ibm	san_jose	0.99
emirates_airlines	dubai	0.93
honda	torrance	0.93
horizon	seattle	0.93
egyptair	cairo	0.93
adobe	san_jose	0.93
...	...	...

```
select x.Product, x.Company
from ProducesProduct x, HeadquarteredIn y
where x.Company=y.Company and
y.City='san_jose'
```

0.9x0.93=0.83

Product	Company	P
personal_computer	ibm	0.95
adobe_indesign	adobe	0.83
adobe_dreamweaver	adobe	0.80

[Example from Suciu et al 2011]

Querying: probabilistic db

ProducesProduct

Company	Product	P
sony	walkman	0.96
microsoft	mac_os_x	0.96
ibm	personal_computer	0.96
microsoft	mac_os	0.9
adobe	adobe_indesign	0.9
adobe	adobe_dreamweaver	0.87
...	...	...

HeadquarteredIn

Company	City	P
microsoft	redmond	1.00
ibm	san_jose	0.99
emirates_airlines	dubai	0.93
honda	torrance	0.93
horizon	seattle	0.93
egyptair	cairo	0.93
adobe	san_jose	0.93
...	...	...

```
select x.Product, x.Company
from ProducesProduct x, HeadquarteredIn y
where x.Company=y.Company and
y.City='san_jose'
```

Product	Company	P
personal_computer	ibm	0.95
adobe_indesign	adobe	0.83
adobe_dreamweaver	adobe	0.80

0.87x0.93=0.80

[Example from Suciu et al 2011]

Querying: probabilistic db

ProducesProduct

Company	Product	P
sony	walkman	0.96
microsoft	mac_os_x	0.96
ibm	personal_computer	0.96
microsoft	mac_os	0.9
adobe	adobe_indesign	0.9
adobe	adobe_dreamweaver	0.87
...	...	...

HeadquarteredIn

Company	City	P
microsoft	redmond	1.00
ibm	san_jose	0.99
emirates_airlines	dubai	0.93
honda	torrance	0.93
horizon	seattle	0.93
egyptair	cairo	0.93
adobe	san_jose	0.93
...	...	...

```
select x.Product, x.Company
from ProducesProduct x, HeadquarteredIn y
where x.Company=y.Company and
y.City='san_jose'
```

answer tuples ranked by probability

Product	Company	P
personal_computer	ibm	0.95
adobe_indesign	adobe	0.83
adobe_dreamweaver	adobe	0.80

[Example from Suciu et al 2011]

PDB with tuple-level uncertainty in ProbLog?

ProducesProduct

Company	Product	P
sony	walkman	0.96
microsoft	mac_os_x	0.96
ibm	personal_computer	0.96
microsoft	mac_os	0.9
adobe	adobe_indesign	0.9
adobe	adobe_dreamweaver	0.87
...	...	...

HeadquarteredIn

Company	City	P
microsoft	redmond	1.00
ibm	san_jose	0.99
emirates_airlines	dubai	0.93
honda	torrance	0.93
horizon	seattle	0.93
egyptair	cairo	0.93
adobe	san_jose	0.93
...	...	...

```
select x.Product, x.Company
from ProducesProduct x, HeadquarteredIn y
where x.Company=y.Company and
y.City='san_jose'
```


[Example from Suciu et al 2011]

PDB with tuple-level uncertainty in ProbLog?

ProducesProduct

Company	Product	P
sony	walkman	0.96
microsoft	mac_os_x	0.96
ibm	personal_computer	0.96
microsoft	mac_os	0.9
adobe	adobe_indesign	0.9
adobe	adobe_dreamweaver	0.87
...	...	...

[Example from Suciu et al 2011]

PDB with tuple-level uncertainty in ProbLog?

ProducesProduct

Company	Product	P
sony	walkman	0.96
microsoft	mac_os_x	0.96
ibm	personal_computer	0.96
microsoft	mac_os	0.9
adobe	adobe_indesign	0.9
adobe	adobe_dreamweaver	0.87
...	...	...

```
0.96::producesProduct(sony,walkman).
0.96::producesProduct(microsoft,mac_os_x).
0.96::producesProduct(ibm,personal_computer).
0.9::producesProduct(microsoft,mac_os).
0.9::producesProduct(adobe,adobe_indesign).
0.87::producesProduct(adobe,adobe_dreamweaver).
...
```


[Example from Suciu et al 2011]

PDB with tuple-level uncertainty in ProbLog?

```
select x.Product, x.Company
from ProducesProduct x, HeadquarteredIn y
where x.Company=y.Company and y.City='san_jose'
```

PDB with tuple-level uncertainty in ProbLog?

```
select x.Product, x.Company
from ProducesProduct x, HeadquarteredIn y
where x.Company=y.Company and y.City='san_jose'
```

```
result(Product,Company) :-
    producesProduct(Company,Product),
    headquarteredIn(Company,san_jose).
query(result(_,_)).
```

PDB with tuple-level uncertainty in ProbLog?

```
select x.Product, x.Company
from ProducesProduct x, HeadquarteredIn y
where x.Company=y.Company and y.City='san_jose'
```

```
result(Product,Company) :-
    producesProduct(Company,Product),
    headquarteredIn(Company,san_jose).
query(result(_,_)).
```

PDB with tuple-level uncertainty in ProbLog?

```
0.96::producesProduct(sony,walkman).
0.96::producesProduct(microsoft,mac_os_x).
0.96::producesProduct(ibm,personal_computer).
0.9::producesProduct(microsoft,mac_os).
0.9::producesProduct(adobe,adobe_indesign).
0.87::producesProduct(adobe,adobe_dreamweaver).
...
1.00::headquarteredIn(microsoft,redmond).
0.99::headquarteredIn(ibm,san_jose).
0.93::headquarteredIn(emirates_airlines,dubai).
0.93::headquarteredIn(honda,torrance).
0.93::headquarteredIn(horizon,seattle).
0.93::headquarteredIn(egyptair,cairo).
0.93::headquarteredIn(adobe,san_jose).
...
      result(Product,Company) :- producesProduct(Company,Product),
                                 headquarteredIn(Company,san_jose).

      query(result(_,_)).
```

PDB with attribute-level uncertainty in ProbLog?

color

item	color	P
mug	green blue	0.65 0.35
plate	pink red purple	0.23 0.14 0.63

PDB with attribute-level uncertainty in ProbLog?

color

item	color	P
mug	green blue	0.65 0.35
plate	pink red purple	0.23 0.14 0.63

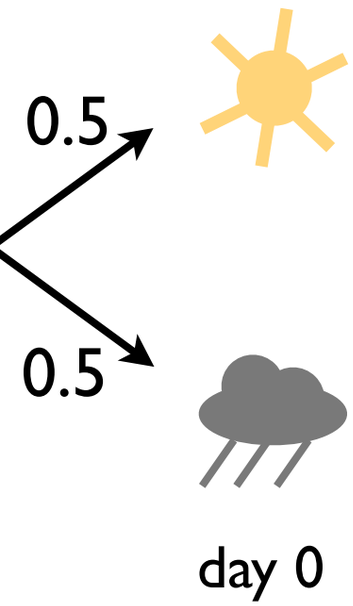
```
0.65::color(mug,green); 0.35::color(mug,blue) <- true.
0.23::color(plate,pink); 0.14::color(plate,red);
                                0.63::color(plate,purple) <- true.
```

ProbLog by example:

Rain or sun?

ProbLog by example:

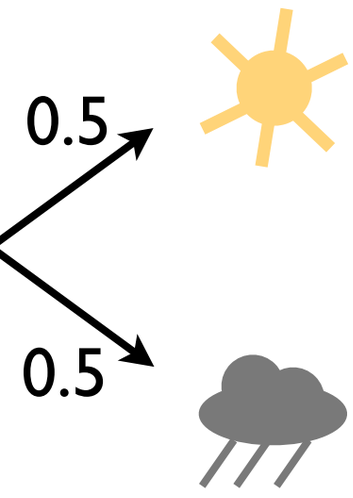
Rain or sun?



day 0

ProbLog by example:

Rain or sun?

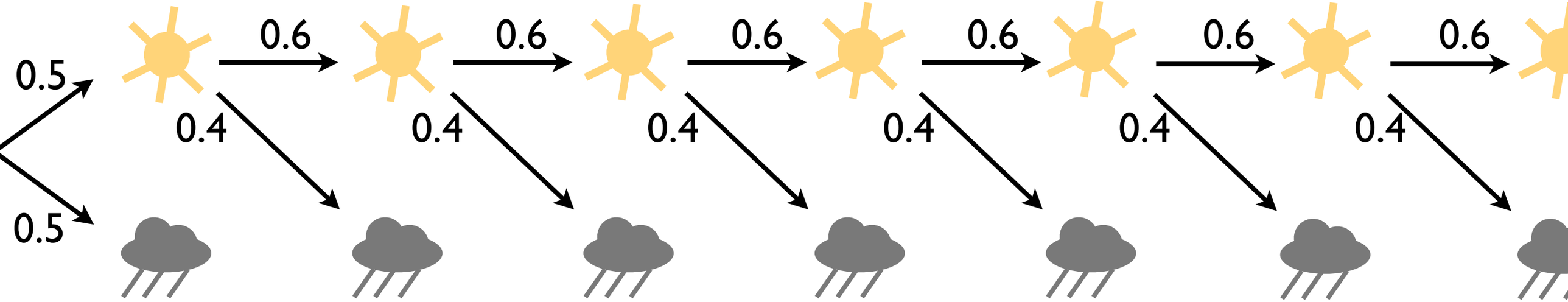


day 0

```
0.5::weather(sun,0) ; 0.5::weather(rain,0) <- true.
```

ProbLog by example:

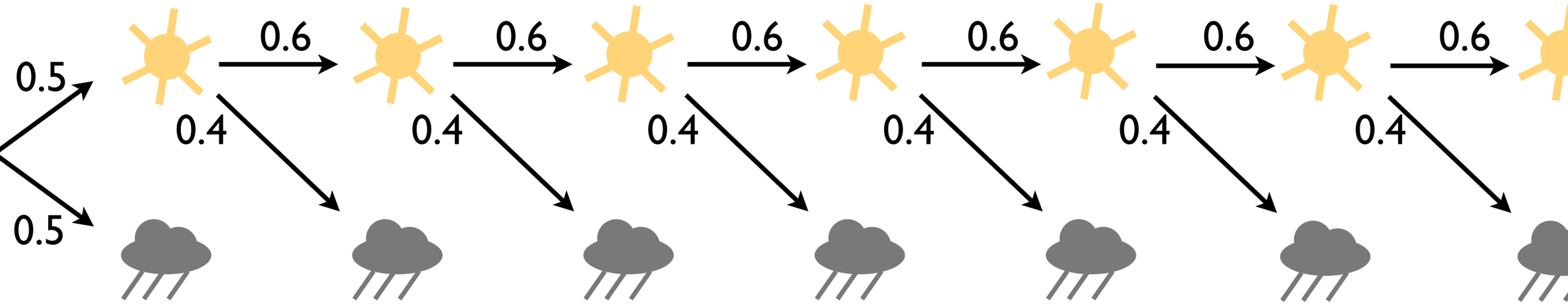
Rain or sun?




```
0.5::weather(sun,0) ; 0.5::weather(rain,0) <- true.
```

ProbLog by example:

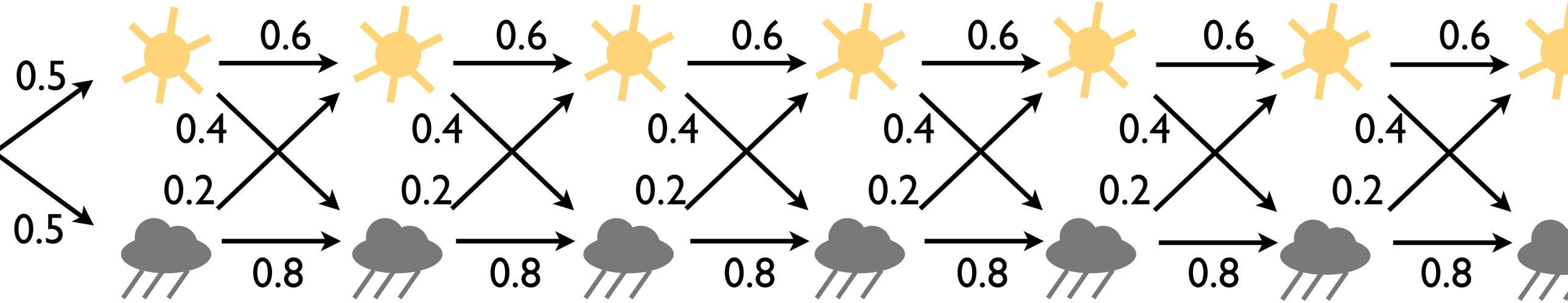
Rain or sun?




```
0.5::weather(sun,0) ; 0.5::weather(rain,0) <- true.

0.6::weather(sun,T) ; 0.4::weather(rain,T)
                    <- T>0, Tprev is T-1, weather(sun,Tprev).
```

ProbLog by example:

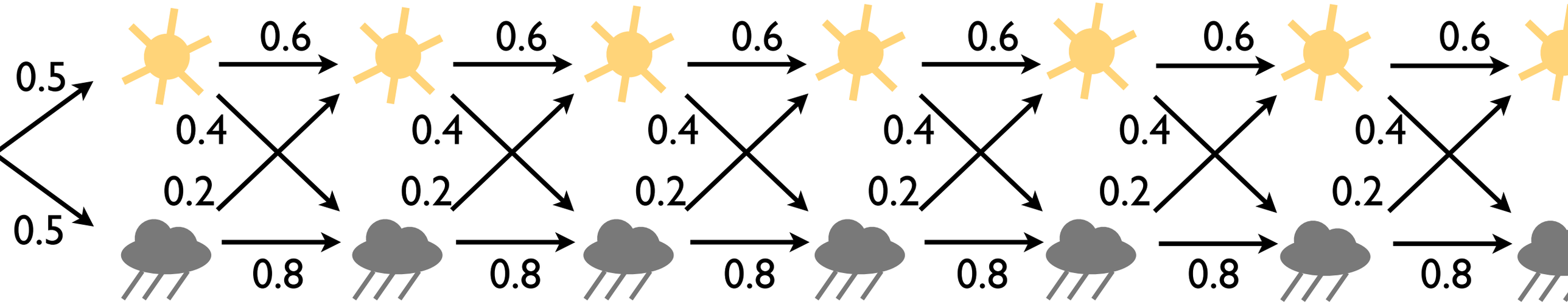
Rain or sun?




```
0.5::weather(sun,0) ; 0.5::weather(rain,0) <- true.

0.6::weather(sun,T) ; 0.4::weather(rain,T)
                  <- T>0, Tprev is T-1, weather(sun,Tprev).
```

ProbLog by example:

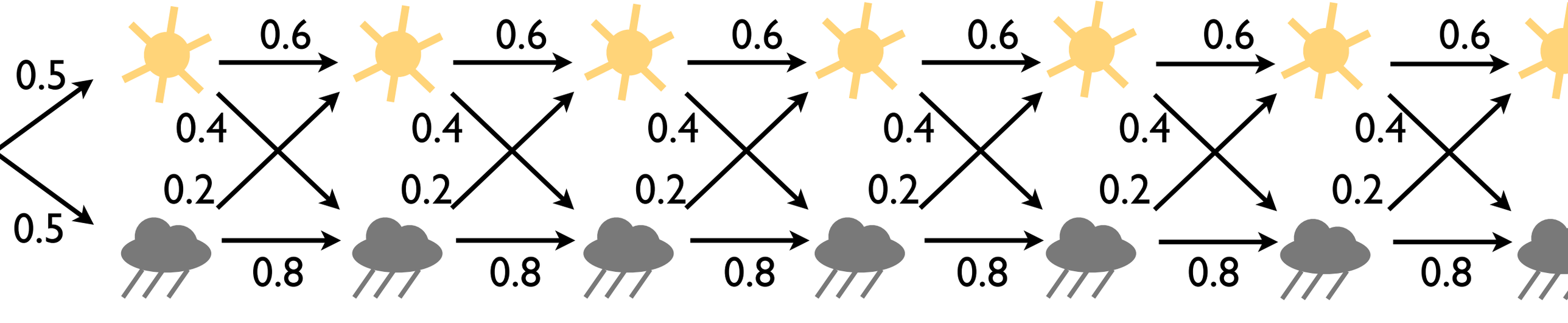
Rain or sun?




```
0.5::weather(sun,0) ; 0.5::weather(rain,0) <- true.

0.6::weather(sun,T) ; 0.4::weather(rain,T)
                  <- T>0, Tprev is T-1, weather(sun,Tprev).
0.2::weather(sun,T) ; 0.8::weather(rain,T)
                  <- T>0, Tprev is T-1, weather(rain,Tprev).
```

ProbLog by example:

Rain or sun?

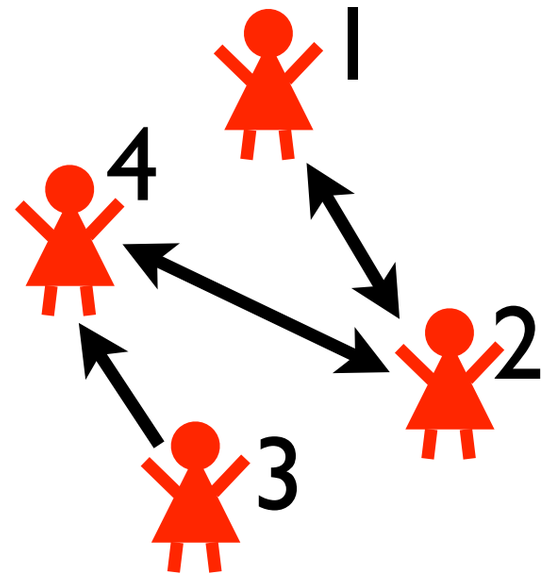



```
0.5::weather(sun,0) ; 0.5::weather(rain,0) <- true.

0.6::weather(sun,T) ; 0.4::weather(rain,T)
                  <- T>0, Tprev is T-1, weather(sun,Tprev).
0.2::weather(sun,T) ; 0.8::weather(rain,T)
                  <- T>0, Tprev is T-1, weather(rain,Tprev).
```

infinite possible worlds! BUT: finitely many partial worlds suffice to answer any given ground query

ProbLog by example:

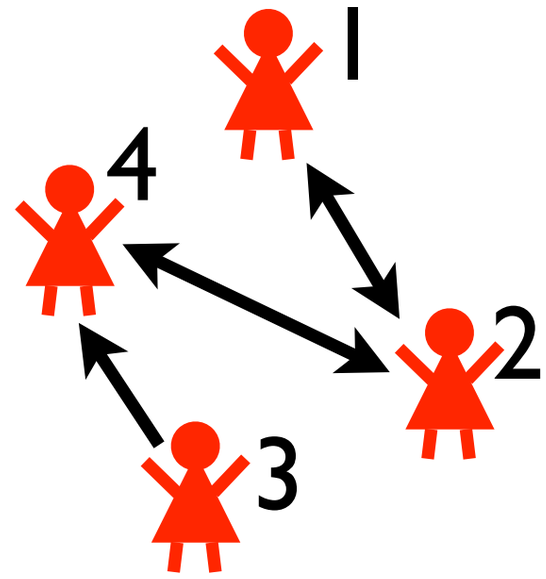
Friends & smokers




```
person(1).
person(2).
person(3).
person(4).

friend(1,2).
friend(2,1).
friend(2,4).
friend(3,4).
friend(4,2).
```

ProbLog by example:

Friends & smokers



typed probabilistic facts

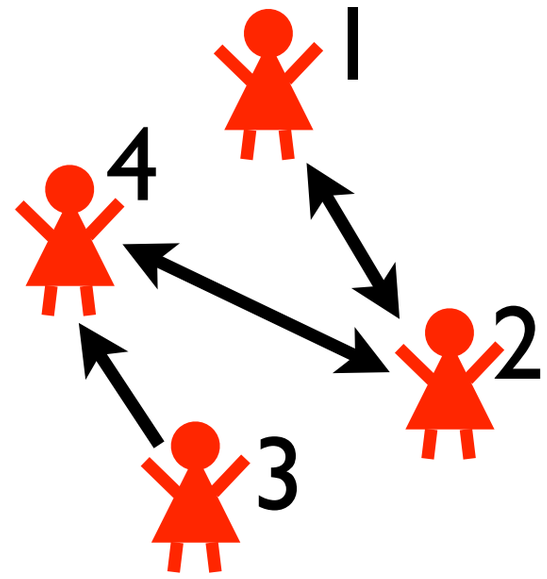
= a probabilistic fact for each grounding

```
0.3::stress(X):- person(X).
0.2::influences(X,Y):-
                person(X), person(Y).
```

```
person(1).
person(2).
person(3).
person(4).

friend(1,2).
friend(2,1).
friend(2,4).
friend(3,4).
friend(4,2).
```

ProbLog by example:

Friends & smokers



typed probabilistic facts

= a probabilistic fact for each grounding

```
0.3::stress(X):- person(X).
0.2::influences(X,Y):-
              person(X), person(Y).
```

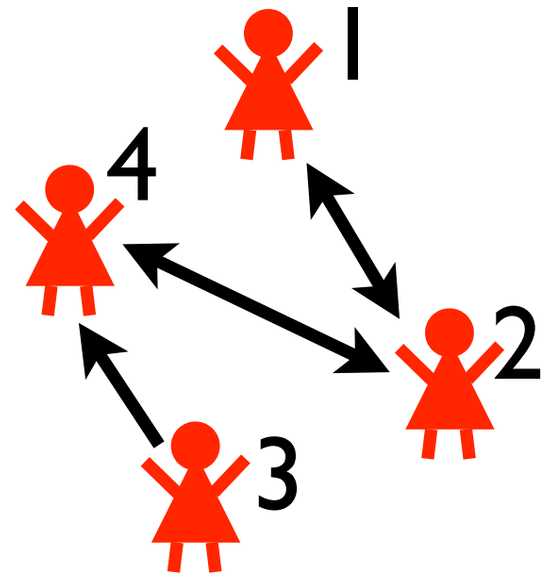
```
0.3::stress(1).
0.3::stress(2).
0.3::stress(3).
0.3::stress(4).
```

```
0.2::influences(1,1).
0.2::influences(1,2).
0.2::influences(1,3).
0.2::influences(1,4).
0.2::influences(2,1).
...
0.2::influences(4,2).
0.2::influences(4,3).
0.2::influences(4,4).
```

```
person(1).
person(2).
person(3).
person(4).

friend(1,2).
friend(2,1).
friend(2,4).
friend(3,4).
friend(4,2).
```

ProbLog by example:

Friends & smokers

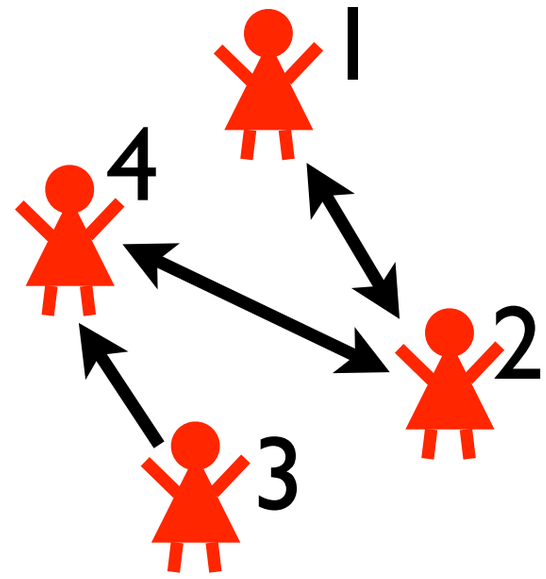



```
0.3::stress(X):- person(X).
0.2::influences(X,Y):-
              person(X), person(Y).

smokes(X) :- stress(X).
smokes(X) :-
     friend(X,Y), influences(Y,X), smokes(Y).
```

```
person(1).
person(2).
person(3).
person(4).

friend(1,2).
friend(2,1).
friend(2,4).
friend(3,4).
friend(4,2).
```

ProbLog by example:

Friends & smokers




```
0.3::stress(X):- person(X).
0.2::influences(X,Y):-
              person(X), person(Y).

smokes(X)  :- stress(X).
smokes(X)  :-
     friend(X,Y), influences(Y,X), smokes(Y).

0.4::asthma(X) <- smokes(X).
```

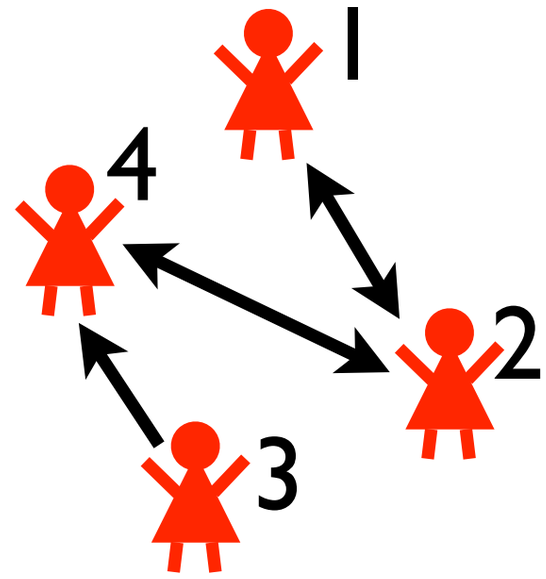
```
person(1).
person(2).
person(3).
person(4).

friend(1,2).
friend(2,1).
friend(2,4).
friend(3,4).
friend(4,2).
```

annotated disjunction with implicit head atom:

with probability 0.6, nothing happens

ProbLog by example:

Friends & smokers




```
0.3::stress(X):- person(X).
0.2::influences(X,Y):-
              person(X), person(Y).

smokes(X) :- stress(X).
smokes(X) :-
     friend(X,Y), influences(Y,X), smokes(Y).

0.4::asthma(X) <- smokes(X).
```

```
person(1).
person(2).
person(3).
person(4).

friend(1,2).
friend(2,1).
friend(2,4).
friend(3,4).
friend(4,2).
```

ProbLog by example:

Limited Luggage

```
weight(skis,6).
weight(boots,4).
weight(helmet,3).
weight(gloves,2).
```

ProbLog by example:

Limited Luggage

```
weight(skis,6).
weight(boots,4).
weight(helmet,3).
weight(gloves,2).

P::pack(Item) :- weight(Item,Weight), P is 1.0/Weight.
```

flexible probability:
computed from the weight of the item

ProbLog by example:

Limited Luggage

```
weight(skis,6).
weight(boots,4).
weight(helmet,3).
weight(gloves,2).
```

```
1/6::pack(skis).
1/4::pack(boots).
1/3::pack(helmet).
1/2::pack(gloves).
```

```
P::pack(Item) :- weight(Item,Weight), P is 1.0/Weight.
```

flexible probability:
computed from the weight of the item

ProbLog by example:

Limited Luggage

```
weight(skis,6).
weight(boots,4).
weight(helmet,3).
weight(gloves,2).

P::pack(Item) :- weight(Item,Weight),  P is 1.0/Weight.

excess(Limit) :- excess([skis,boots,helmet,gloves],Limit).
```

list of all items

ProbLog by example:

Limited Luggage

```
weight(skis,6).
weight(boots,4).
weight(helmet,3).
weight(gloves,2).

P::pack(Item) :- weight(Item,Weight),  P is 1.0/Weight.

excess(Limit) :- excess([skis,boots,helmet,gloves],Limit).

excess([],Limit) :- Limit<0.
excess([I|R],Limit) :-
   pack(I), weight(I,W), L is Limit-W, excess(R,L).
excess([I|R],Limit) :-
   \+pack(I), excess(R,Limit).
```

ProbLog by example:

Limited Luggage

```
weight(skis,6).
weight(boots,4).
weight(helmet,3).
weight(gloves,2).

P::pack(Item) :- weight(Item,Weight),  P is 1.0/Weight.

excess(Limit) :- excess([skis,boots,helmet,gloves],Limit).

excess([],Limit) :- Limit<0.
excess([I|R],Limit) :-
   pack(I), weight(I,W), L is Limit-W, excess(R,L).
excess([I|R],Limit) :-
   \+pack(I), excess(R,Limit).
```

pack first item, decrease limit by its weight, and continue with rest of items

ProbLog by example:

Limited Luggage

```
weight(skis,6).
weight(boots,4).
weight(helmet,3).
weight(gloves,2).

P::pack(Item) :- weight(Item,Weight),  P is 1.0/Weight.

excess(Limit) :- excess([skis,boots,helmet,gloves],Limit).

excess([],Limit) :- Limit<0.
excess([I|R],Limit) :-
   pack(I), weight(I,W), L is Limit-W, excess(R,L).
excess([I|R],Limit) :-
   \+pack(I), excess(R,Limit).
```

do **not** pack first item,
continue with rest of items

ProbLog by example:

Limited Luggage

```
weight(skis,6).
weight(boots,4).
weight(helmet,3).
weight(gloves,2).

P::pack(Item) :- weight(Item,Weight),  P is 1.0/Weight.

excess(Limit) :- excess([skis,boots,helmet,gloves],Limit).

excess([],Limit) :- Limit<0.
excess([I|R],Limit) :-
   pack(I), weight(I,W), L is Limit-W, excess(R,L).
excess([I|R],Limit) :-
   \+pack(I), excess(R,Limit).
```

no items left: did we add too much?

ProbLog by example:

Limited Luggage

```
weight(skis,6).
weight(boots,4).
weight(helmet,3).
weight(gloves,2).

P::pack(Item) :- weight(Item,Weight),  P is 1.0/Weight.

excess(Limit) :- excess([skis,boots,helmet,gloves],Limit).

excess([],Limit) :- Limit<0.
excess([I|R],Limit) :-
   pack(I), weight(I,W), L is Limit-W, excess(R,L).
excess([I|R],Limit) :-
   \+pack(I), excess(R,Limit).
```

Summary: ProbLog Syntax

- input database: ground facts

```
person(bob).
```

- probabilistic facts

```
0.5::stress(bob).
```

- typed probabilistic facts (body deterministic)

```
0.5::stress(X) :- person(X).
```

- flexible probabilities

```
P::pack(Item) :- weight(Item,W),
                 P is 1.0/W.
```

- annotated disjunctions

```
0.4::asthma(X) <- smokes(X).
```

```
0.5::weather(sun,0) ; 0.5::weather(rain,0) <- true.
```

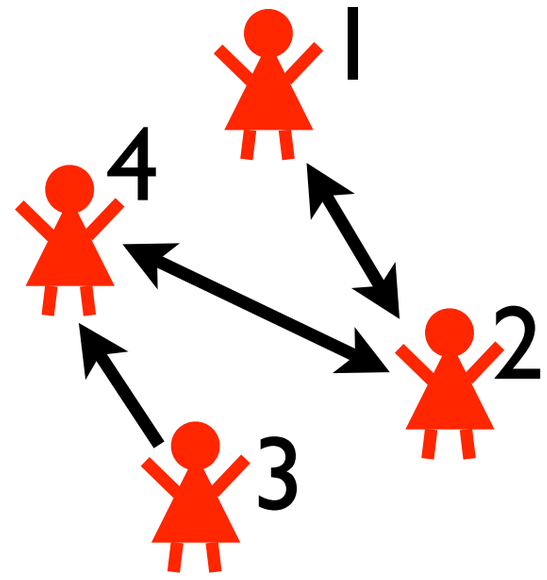
- Prolog clauses

```
smokes(X) :- influences(Y,X), smokes(Y).
```

```
excess([I|R],Limit) :- \+pack(I), excess(R,Limit).
```

ProbLog example:

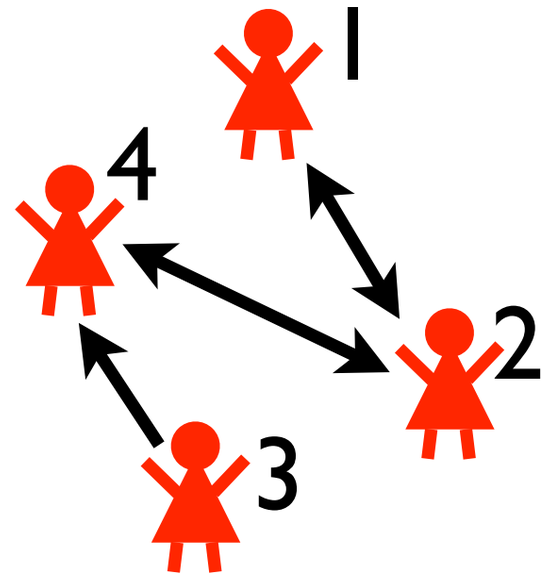
Friends & smokers




```
0.5::stress(1).
0.1::stress(2).
0.8::stress(3).
0.3::stress(4).

0.9::friend(1,2).
0.8::friend(2,1).
0.3::friend(2,4).
0.7::friend(3,4).
0.1::friend(4,2).

smokes(X) :- stress(X).
smokes(X) :-
     friend(X,Y), smokes(Y).
```

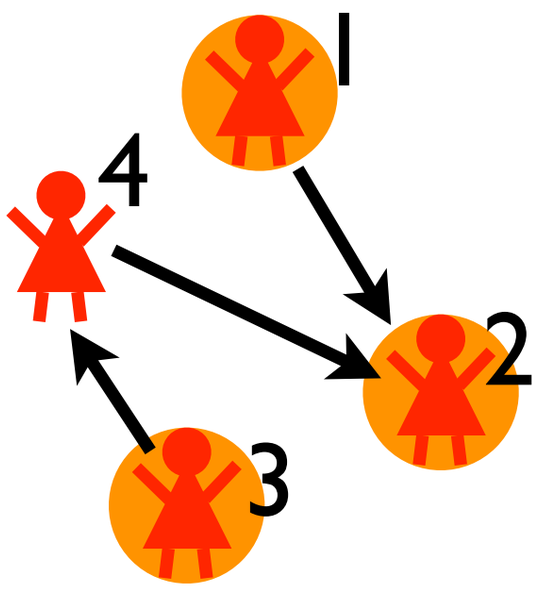
ProbLog example:

Friends & smokers




```
0.5::stress(1).
0.1::stress(2).
0.8::stress(3).
0.3::stress(4).

0.9::friend(1,2).
0.8::friend(2,1).
0.3::friend(2,4).
0.7::friend(3,4).
0.1::friend(4,2).

smokes(X) :- stress(X).
smokes(X) :-
     friend(X,Y), smokes(Y).
```

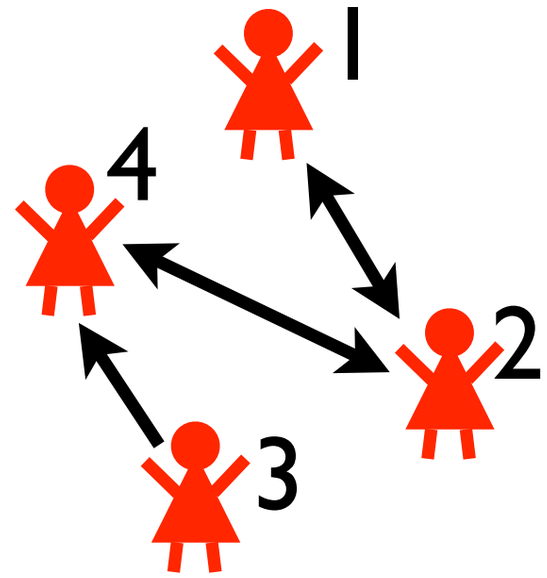
example possible world




```
friend(2,1).
friend(3,4).
friend(2,4).
stress(1).
stress(2).
stress(3).
smokes(1).
smokes(2).
smokes(3).
```

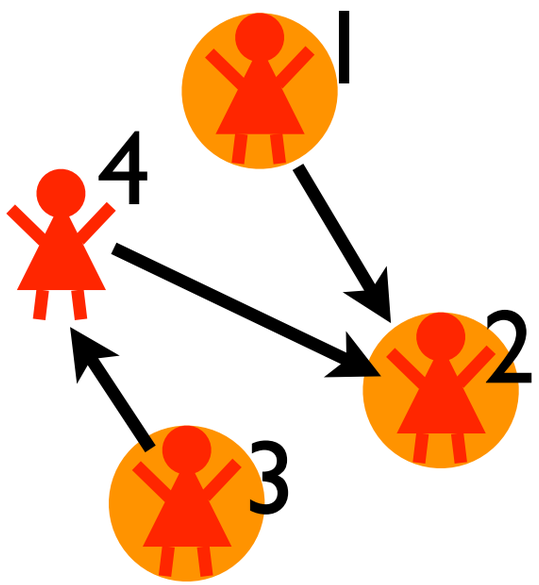
ProbLog example:

Friends & smokers




```
0.5::stress(1).
0.1::stress(2).
0.8::stress(3).
0.3::stress(4).

0.9::friend(1,2).
0.8::friend(2,1).
0.3::friend(2,4).
0.7::friend(3,4).
0.1::friend(4,2).

smokes(X) :- stress(X).
smokes(X) :-
     friend(X,Y), smokes(Y).
```

example possible world

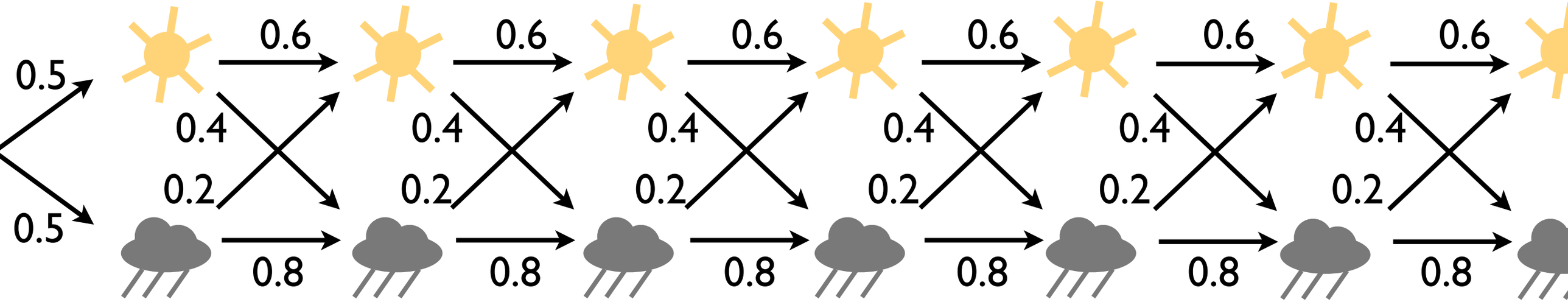



```
friend(2,1).
friend(3,4).
friend(2,4).
stress(1).
stress(2).
stress(3).
smokes(1).
smokes(2).
smokes(3).
```

- several instances of `smokes(X)` in same world
- `smokes(2)` : multiple derivations in same world
- distribution over worlds not (always) a distribution over computations / answers

ProbLog by example:

Rain or sun?



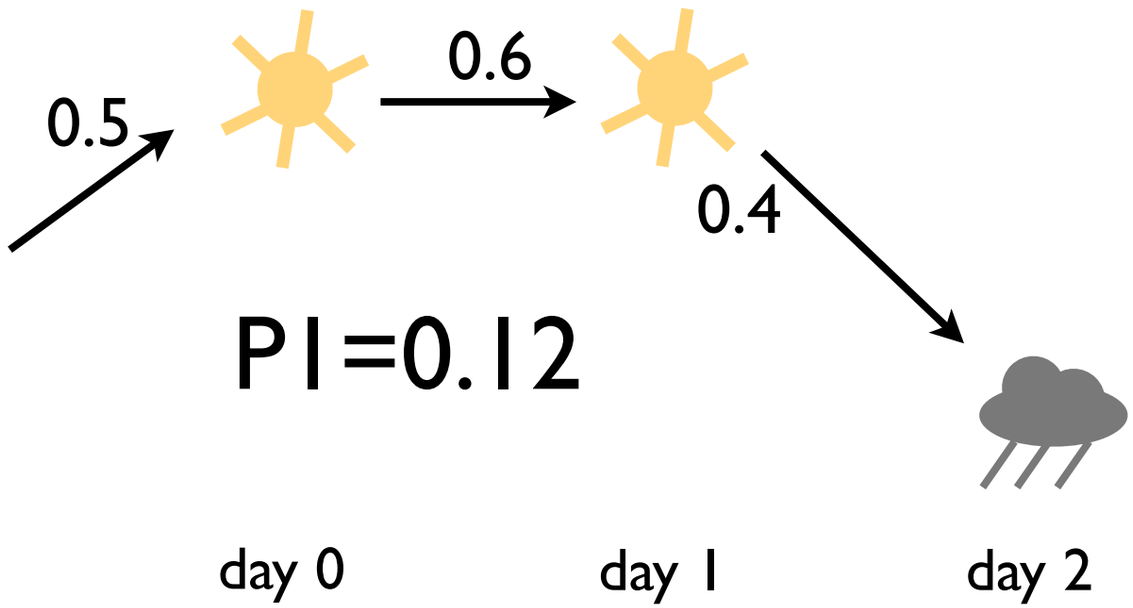

```
0.5::weather(sun,0) ; 0.5::weather(rain,0) <- true.

0.6::weather(sun,T) ; 0.4::weather(rain,T)
                    <- T>0, Tprev is T-1, weather(sun,Tprev).
0.2::weather(sun,T) ; 0.8::weather(rain,T)
                    <- T>0, Tprev is T-1, weather(rain,Tprev).
```

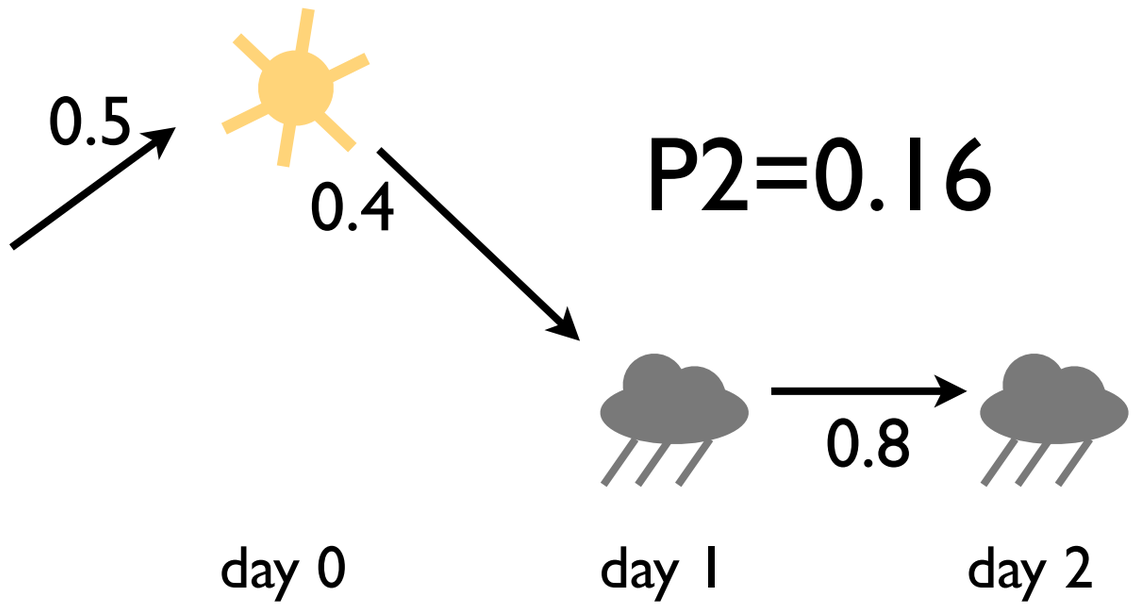
≤1 proof for a ground query per possible world → distribution over worlds is distribution over derivations!

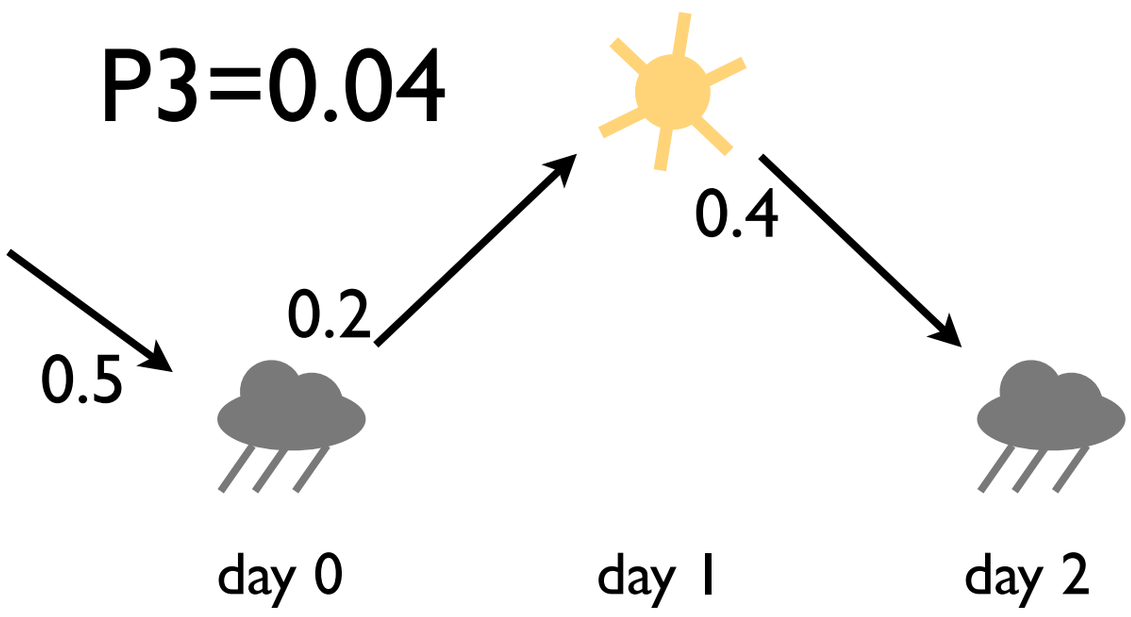
Possible worlds

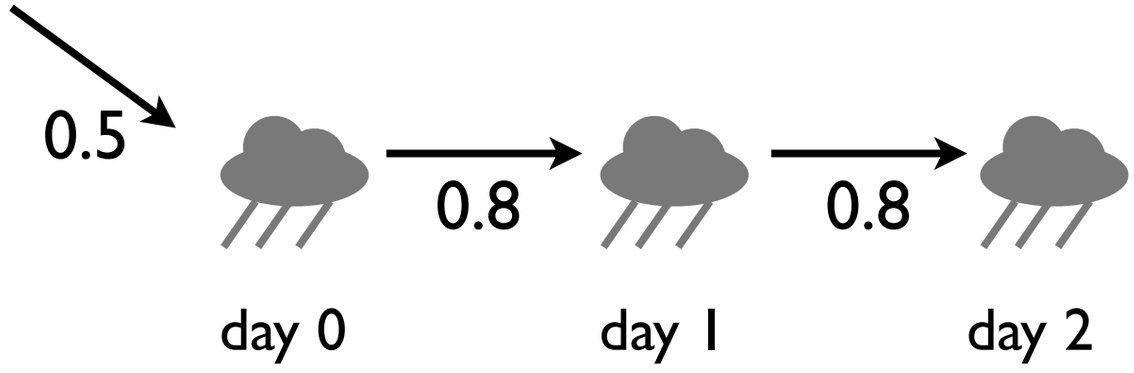
```
?- weather(rain,2).
```

P=P1+P2+P3+P4









Mutually Exclusive Rules:

no two rules apply simultaneously

```
0.5::weather(sun,0) ; 0.5::weather(rain,0) <- true.

0.6::weather(sun,T) ; 0.4::weather(rain,T)
                <- T>0, Tprev is T-1, weather(sun,Tprev).

0.2::weather(sun,T) ; 0.8::weather(rain,T)
                <- T>0, Tprev is T-1, weather(rain,Tprev).
```

Mutually Exclusive Rules:

no two rules apply simultaneously

first rule for day 0, others for later days

```
0.5::weather(sun,0) ; 0.5::weather(rain,0) <- true.

0.6::weather(sun,T) ; 0.4::weather(rain,T)
                <- T>0, Tprev is T-1, weather(sun,Tprev).

0.2::weather(sun,T) ; 0.8::weather(rain,T)
                <- T>0, Tprev is T-1, weather(rain,Tprev).
```

Mutually Exclusive Rules:

no two rules apply simultaneously

first rule for day 0, others for later days

day 0: either sun or rain

```
0.5::weather(sun,0) ; 0.5::weather(rain,0) <- true.

0.6::weather(sun,T) ; 0.4::weather(rain,T)
                <- T>0, Tprev is T-1, weather(sun,Tprev).

0.2::weather(sun,T) ; 0.8::weather(rain,T)
                <- T>0, Tprev is T-1, weather(rain,Tprev).
```

Mutually Exclusive Rules:

no two rules apply simultaneously

first rule for day 0, others for later days

day 0: either sun or rain

```
0.5::weather(sun,0) ; 0.5::weather(rain,0) <- true.

0.6::weather(sun,T) ; 0.4::weather(rain,T)
                    <- T>0, Tprev is T-1, weather(sun,Tprev).

0.2::weather(sun,T) ; 0.8::weather(rain,T)
                    <- T>0, Tprev is T-1, weather(rain,Tprev).
```

rules for T>0 cover mutually exclusive cases on previous day

[Sato and Kameya, JAIR 01]

PRISM

- Another probabilistic Prolog based on the distribution semantics
- Mutual exclusiveness assumption
  - allows for efficient inference by dynamic programming, cf. probabilistic grammars
  - but excludes certain models, e.g., smokers
- http://sato-www.cs.titech.ac.jp/prism/

PRISM

- "multi-valued random switches" = annotated disjunctions with body *true*
- stochastic memoization: switch gives fresh result on each call
- Prolog rules
- limited support for negation (compiling away)

Weather in PRISM

```
values(init,[sun,rain]).
values(tr(_),[sun,rain]).

:- set_sw(init,[0.5,0.5]).
:- set_sw(tr(sun),[0.6,0.4]).
:- set_sw(tr(rain),[0.2,0.8]).
```

```
weather(W,Time) :-
  Time >= 0,
  msw(init,W0),
  w(0,Time,W0,W).

w(T,T,W,W).
w(Now,T,WNow,WT) :-
   Now < T,
   msw(tr(WNow),WNext),
   Next is Now+1,
   w(Next,T,WNext,WT).
```

Weather in PRISM

```
values(init,[sun,rain]).
values(tr(_),[sun,rain])
```

random variables and their values

```
:- set_sw(init,[0.5,0.5]).
:- set_sw(tr(sun),[0.6,0.4]).
:- set_sw(tr(rain),[0.2,0.8]).
```

```
weather(W,Time) :-
  Time >= 0,
  msw(init,W0),
  w(0,Time,W0,W).

w(T,T,W,W).
w(Now,T,WNow,WT) :-
   Now < T,
   msw(tr(WNow),WNext),
   Next is Now+1,
   w(Next,T,WNext,WT).
```

Weather in PRISM

```
values(init,[sun,rain]).
values(tr(_),[sun,rain])
```

random variables and their values

```
:- set_sw(init,[0.5,0.5]).
:- set_sw(tr(sun),[0.6,0.4]).
:- set_sw(tr(rain),[0.2,0.8]).
```

probability distributions

```
weather(W,Time) :-
  Time >= 0,
  msw(init,W0),
  w(0,Time,W0,W).

w(T,T,W,W).
w(Now,T,WNow,WT) :-
   Now < T,
   msw(tr(WNow),WNext),
   Next is Now+1,
   w(Next,T,WNext,WT).
```

Weather in PRISM

```
values(init,[sun,rain]).
values(tr(_),[sun,rain])
```

random variables and their values

```
:- set_sw(init,[0.5,0.5]).
:- set_sw(tr(sun),[0.6,0.4]).
:- set_sw(tr(rain),[0.2,0.8]).
```

probability distributions

```
weather(W,Time) :-
   Time >= 0,
   msw(init,W0),
   w(0,Time,W0,W).

w(T,T,W,W).
w(Now,T,WNow,WT) :-
    Now < T,
    msw(tr(WNow),WNext),
    Next is Now+1,
    w(Next,T,WNext,WT).
```

set **W0** to random value of **init**

Weather in PRISM

```
values(init,[sun,rain]).
values(tr(_),[sun,rain])
```

random variables and their values

```
:- set_sw(init,[0.5,0.5]).
:- set_sw(tr(sun),[0.6,0.4]).
:- set_sw(tr(rain),[0.2,0.8]).
```

probability distributions

```
weather(W,Time) :-
   Time >= 0,
   msw(init,W0),
   w(0,Time,W0,W).

w(T,T,W,W).
w(Now,T,WNow,WT) :-
    Now < T,
    msw(tr(WNow),WNext),
    Next is Now+1,
    w(Next,T,WNext,WT).
```

set **`W0`** to random value of **`init`**

set **`WNext`** to random value of **`tr(WNow)`**, using **fresh** value on every call

Weather in PRISM / ProbLog

```
values(init,[sun,rain]).
values(tr(_),[sun,rain]).

:- set_sw(init,[0.5,0.5]).
:- set_sw(tr(sun),[0.6,0.4]).
:- set_sw(tr(rain),[0.2,0.8]).
```

```
weather(W,Time) :-
  Time >= 0,
  msw(init,W0),
  w(0,Time,W0,W).

w(T,T,W,W).
w(Now,T,WNow,WT) :-
    Now < T,
    msw(tr(WNow),WNext),
    Next is Now+1,
    w(Next,T,WNext,WT).
```

```
0.5::init(sun) ;
  0.5::init(rain) <- true.
0.6::tr(T,sun,sun) ;
  0.4::tr(T,sun,rain) <- true.
0.2::tr(T,rain,sun) ;
  0.8::tr(T,rain,rain) <- true.
```

```
weather(W,Time) :-
  Time >= 0,
  init(W0),
  w(0,Time,W0,W).

w(T,T,W,W).
w(Now,T,WNow,WT) :-
    Now < T,
    tr(Now,WNow,WNext),
    Next is Now+1,
    w(Next,T,WNext,WT).
```

ProbLog needs to explicitly use different facts at each call

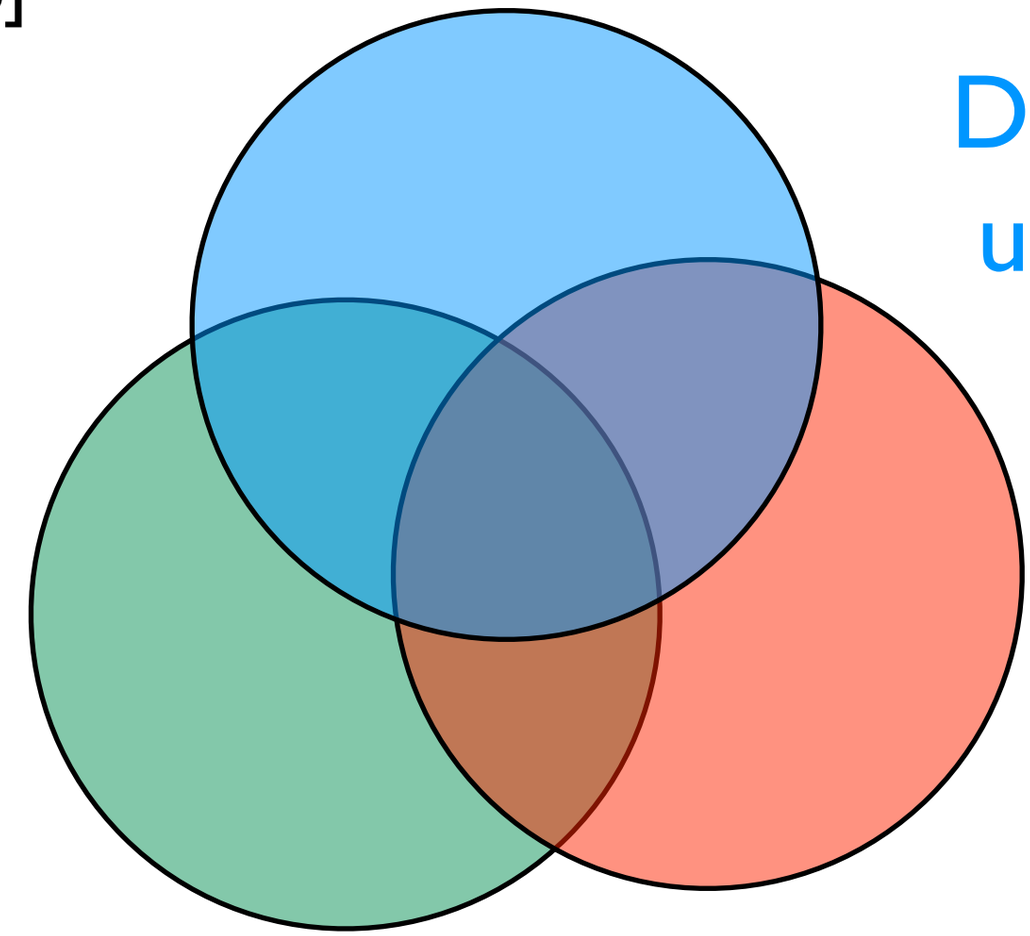
Probabilistic Programming Languages outside LP

- IBAL [Pfeffer 01]
- Figaro [Pfeffer 09]
- Church [Goodman et al 08]
- BLOG [Milch et al 05]
- and many more appearing recently

Church

probabilistic functional programming

[Goodman et al, UAI 08]



Dealing with uncertainty

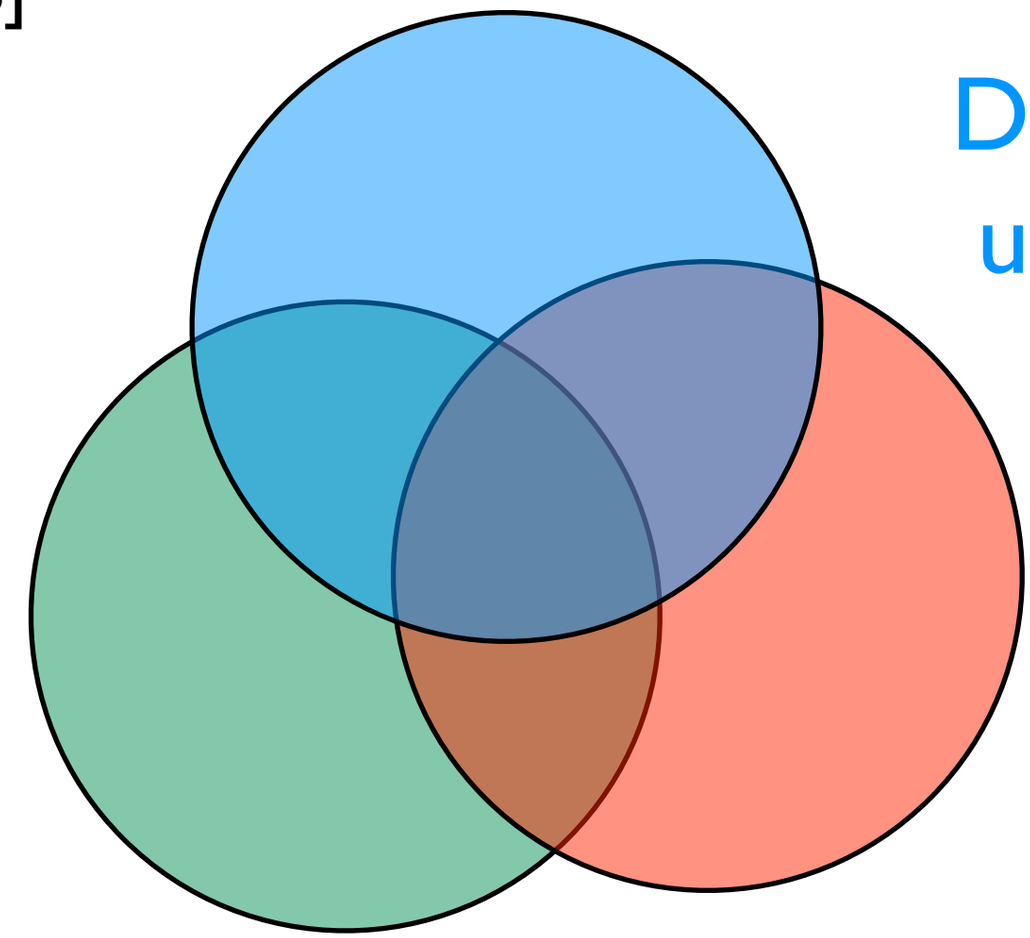
Reasoning with relational data

Learning

Church

probabilistic functional programming

[Goodman et al, UAI 08]



Dealing with uncertainty

functional programming

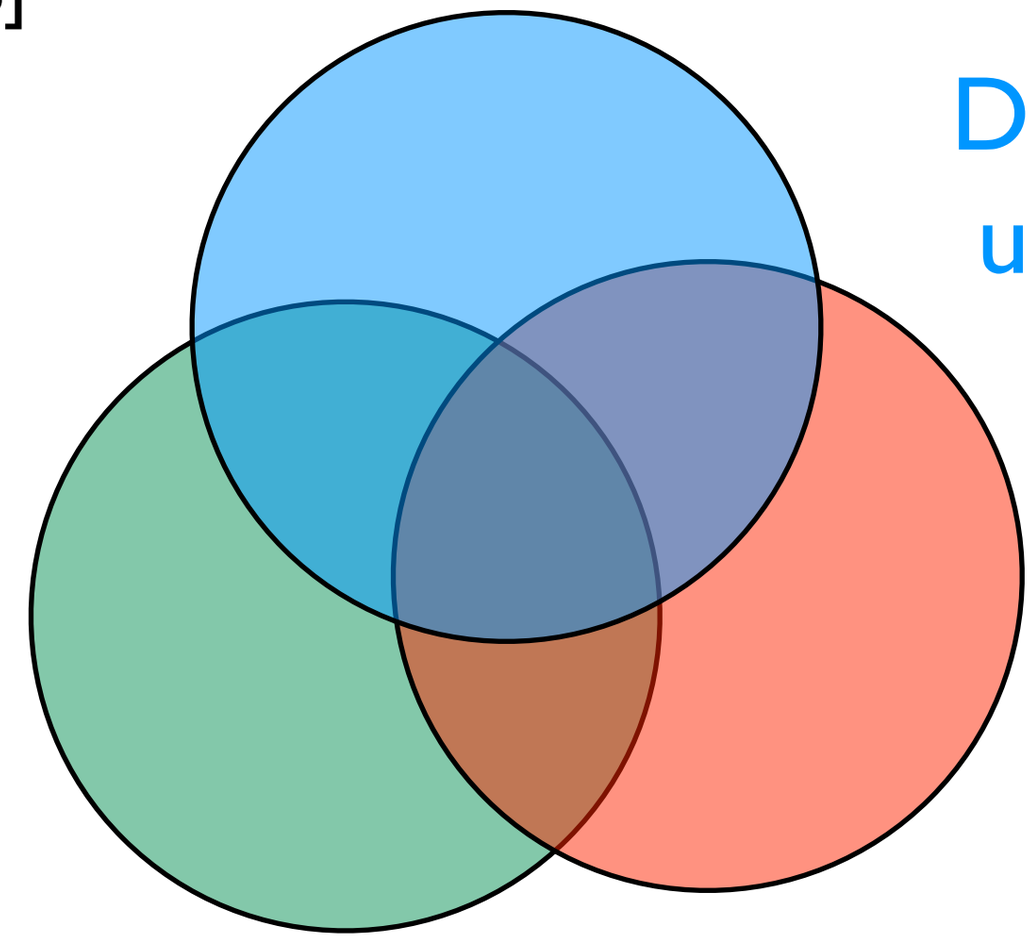
Learning

```
(define plus5 (lambda (x) (+ x 5)))

(map plus5 '(1 2 3))
```

Church

probabilistic functional programming

[Goodman et al, UAI 08]



Dealing with uncertainty

functional programming

Learning

one execution

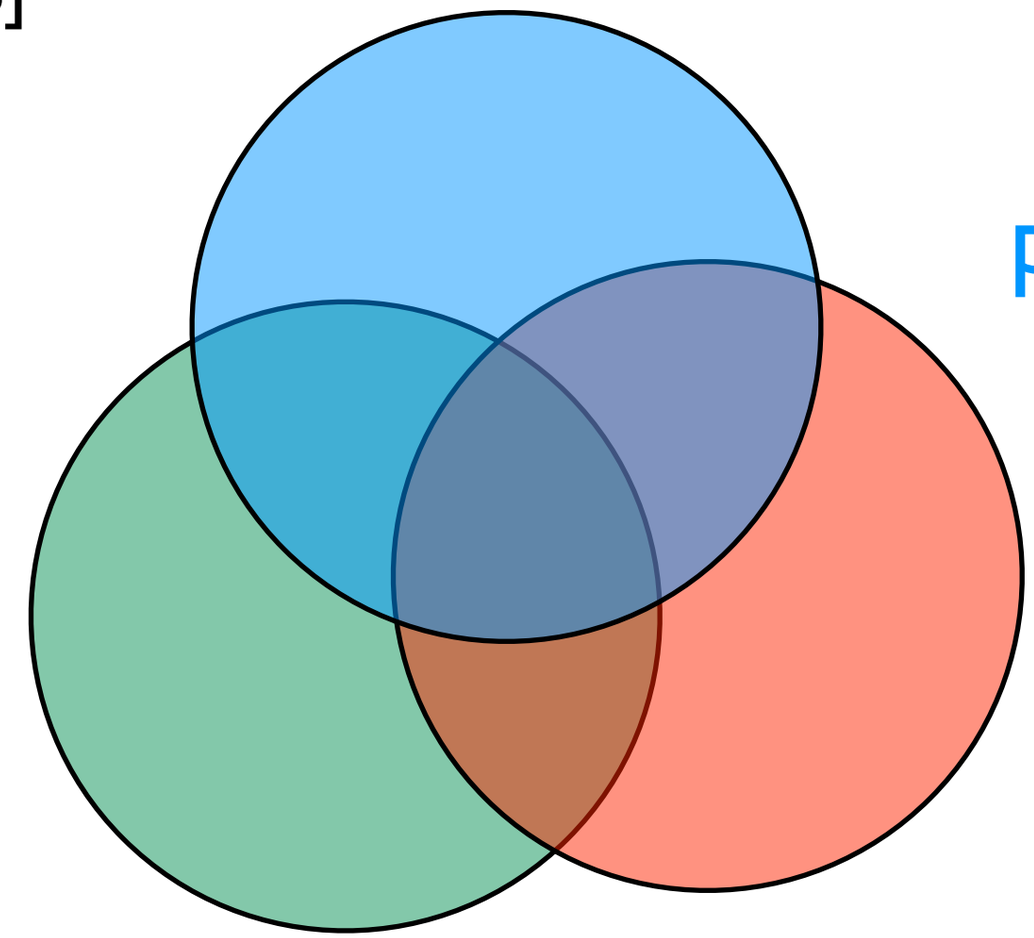
```
(define plus5 (lambda (x) (+ x 5)))

(map plus5 '(1 2 3))
```

Church

probabilistic functional programming

[Goodman et al, UAI 08]

```
(define randplus5
 (lambda (x) (if (flip 0.6)
                 (+ x 5)
                 x)))

(map randplus5 '(1 2 3))
```



random primitives

functional programming

Learning

one execution

```
(define plus5 (lambda (x) (+ x 5)))

(map plus5 '(1 2 3))
```

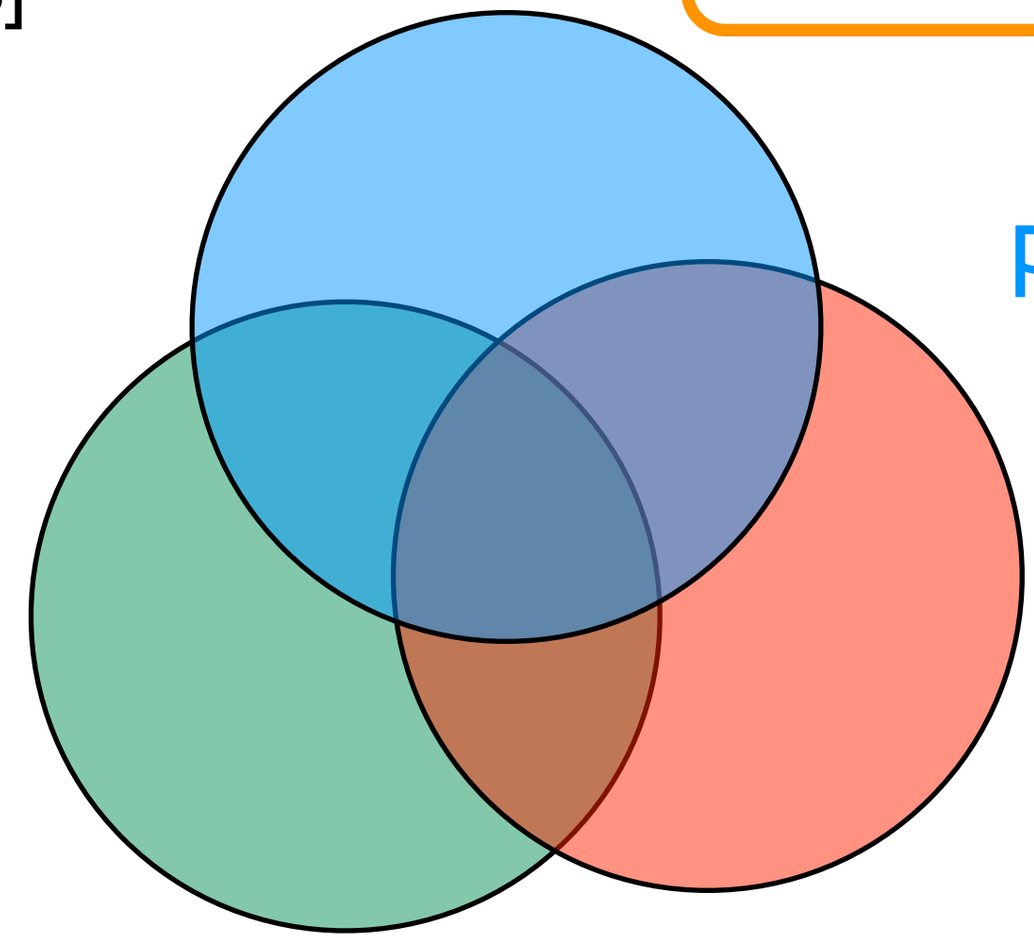
Church

probabilistic functional programming

[Goodman et al, UAI 08]

several possible executions

```
(define randplus5
 (lambda (x) (if (flip 0.6)
                 (+ x 5)
                 x)))

(map randplus5 '(1 2 3))
```



random primitives

functional programming

Learning

one execution

```
(define plus5 (lambda (x) (+ x 5)))

(map plus5 '(1 2 3))
```

Church

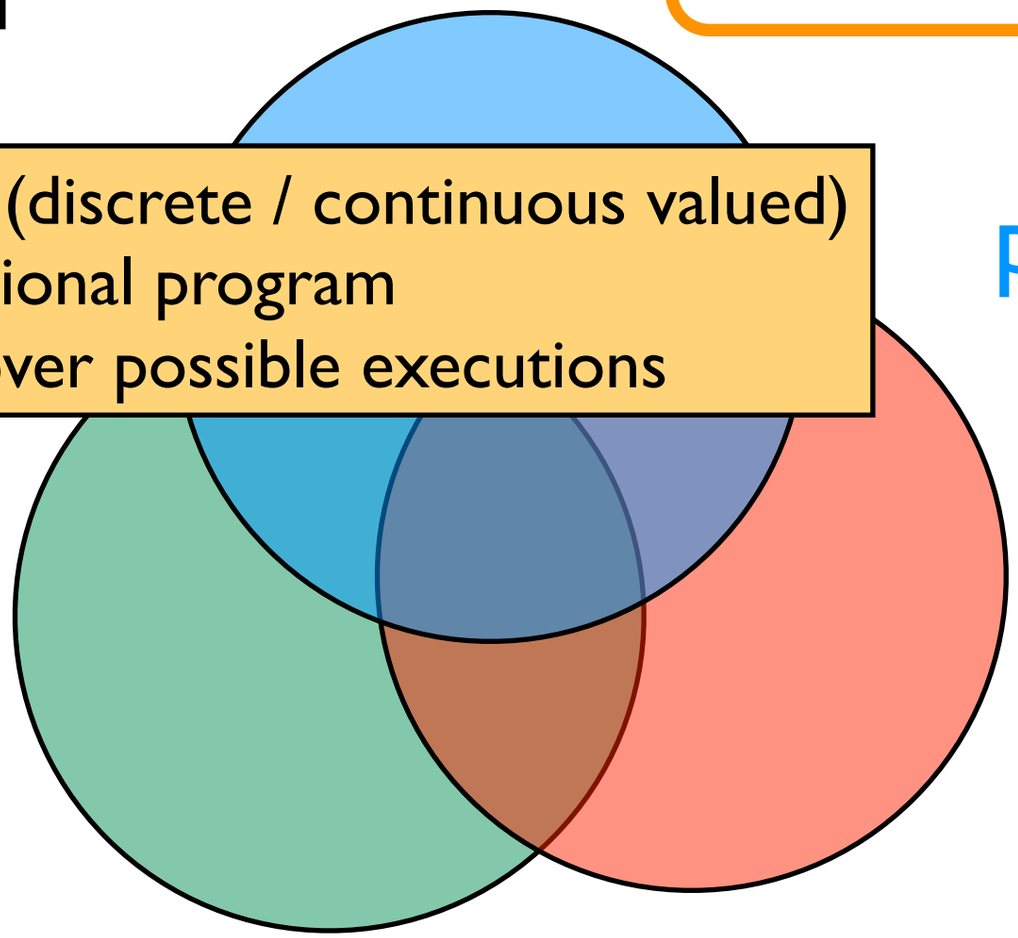
probabilistic functional programming

[Goodman et al, UAI 08]

several possible executions

```
(define randplus5
 (lambda (x) (if (flip 0.6)
                 (+ x 5)
                 x)))

(map randplus5 '(1 2 3))
```

probabilistic primitives (discrete / continuous valued)
+ functional program
→ distribution over possible executions



random primitives

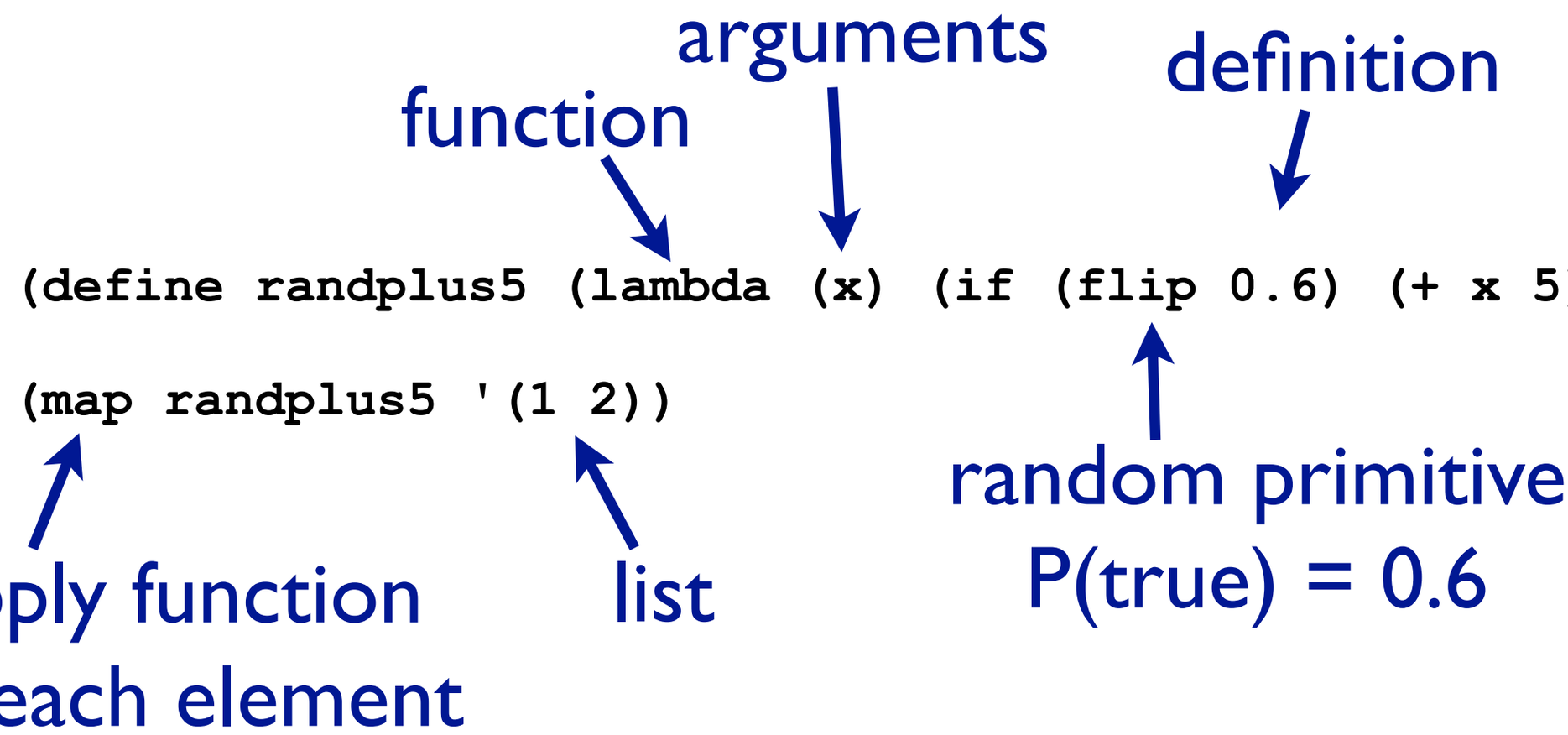
functional programming

Learning

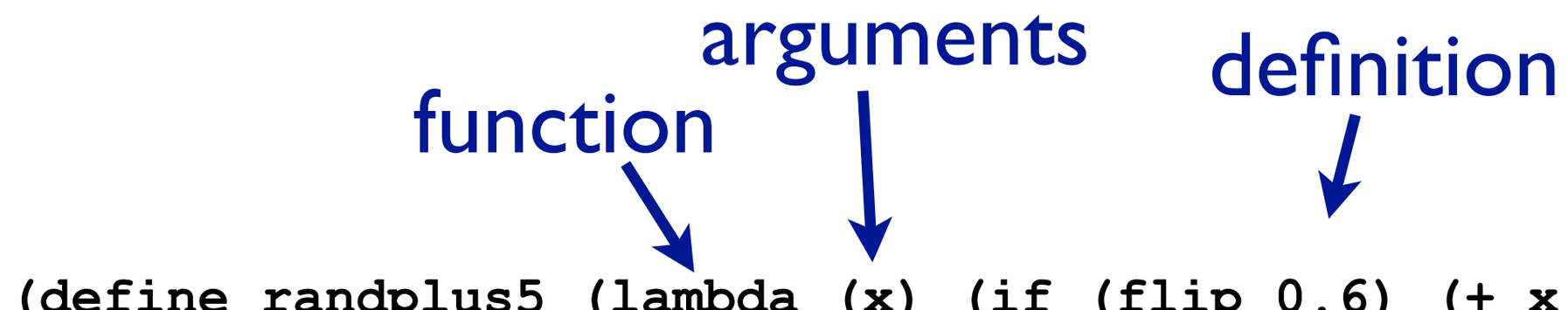
one execution

```
(define plus5 (lambda (x) (+ x 5)))

(map plus5 '(1 2 3))
```

Church Example



Church Example




```
(map randplus5 '(1 2))
```

apply function to each element

list

result: (1 2) with 0.4×0.4
(1 7) with 0.4×0.6
(6 2) with 0.6×0.4
(6 7) with 0.6×0.6

in ProbLog?

```
(define randplus5 (lambda (x) (if (flip 0.6) (+ x 5) x)))

(map randplus5 '(1 2))
```

in ProbLog?

```
(define randplus5 (lambda (x) (if (flip 0.6) (+ x 5) x)))

(map randplus5 '(1 2))
```

```
0.4::p5(N,N);0.6::p5(N,M) <- M is N+5.
lp5([],[]).
lp5([N|L],[M|K]) :-
    p5(N,M),
    lp5(L,K).

query(lp5([1,2],_)).
```

in ProbLog?

```
(define randplus5 (lambda (x) (if (flip 0.6) (+ x 5) x)))

(map randplus5 '(1 2))
```

```
0.4::p5(N,N);0.6::p5(N,M) <- M is N+5.
lp5([],[]).
lp5([N|L],[M|K]) :-
    p5(N,M),
    lp5(L,K).

query(lp5([1,2],_)).
```

result: (1 2) with 0.4×0.4
(1 7) with 0.4×0.6
(6 2) with 0.6×0.4
(6 7) with 0.6×0.6

results for [1,1]?

```
(define randplus5 (lambda (x) (if (flip 0.6) (+ x 5) x)))

(map randplus5 '(1 1))
```

```
0.4::p5(N,N);0.6::p5(N,M) <- M is N+5.
lp5([],[]).
lp5([N|L],[M|K]) :-
    p5(N,M),
    lp5(L,K).

query(lp5([1,1],_)).
```

results for [1,1]?

```
(define randplus5 (lambda (x) (if (flip 0.6) (+ x 5) x)))

(map randplus5 '(1 1))
```

Church result: (1 1) with 0.4×0.4
(1 6) with 0.4×0.6
(6 1) with 0.6×0.4
(6 6) with 0.6×0.6

```
0.4::p5(N,N);0.6::p5(N,M) <- M is N+5.

lp5([],[]).
lp5([N|L],[M|K]) :-
    p5(N,M),
    lp5(L,K).

query(lp5([1,1],_)).
```

results for [1,1]?

```
(define randplus5 (lambda (x) (if (flip 0.6) (+ x 5) x)))

(map randplus5 '(1 1))
```

Church result: (1 1) with 0.4×0.4
(1 6) with 0.4×0.6
(6 1) with 0.6×0.4
(6 6) with 0.6×0.6

```
0.4::p5(N,N);0.6::p5(N,M) <- M is N+5.
lp5([],[]).
lp5([N|L],[M|K]) :-
    p5(N,M),
    lp5(L,K).

query(lp5([1,1],_)).
```

ProbLog result: (1 1) with 0.4
(1 6) with 0.0
(6 1) with 0.0
(6 6) with 0.6

results for [1,1]?

```
(define randplus5 (lambda (x) (if (flip 0.6) (+ x 5) x)))

(map randplus5 '(1 1))
```

Church result: (1 1) with 0.4×0.4
(1 6) with 0.4×0.6
(6 1) with 0.6×0.4
(6 6) with 0.6×0.6

```
0.4::p5(N,N);0.6::p5(N,M) <- M is N+5.

lp5([],[]).
lp5([N|L],[M|K]) :-
    p5(N,M),
    lp5(L,K).

query(lp5([1,1],_)).
```

ProbLog result: (1 1) with 0.4
(1 6) with 0.0
(6 1) with 0.0
(6 6) with 0.6

stochastic memoization

in ProbLog?

```
(define randplus5 (lambda (x) (if (flip 0.6) (+ x 5) x)))

(map randplus5 '(1 2))
```

in ProbLog?

```
(define randplus5 (lambda (x) (if (flip 0.6) (+ x 5) x)))

(map randplus5 '(1 2))
```

```
0.4::p5(N,N,ID);0.6::p5(N,M,ID) <- M is N+5.
lp5([],[]).
lp5([N|L],[M|K]) :-
    p5(N,M,L),
    lp5(L,K).

query(lp5([1,1],_)).
```

in ProbLog?

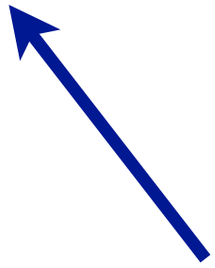
```
(define randplus5 (lambda (x) (if (flip 0.6) (+ x 5) x)))

(map randplus5 '(1 2))
```

```
0.4::p5(N,N,ID);0.6::p5(N,M,ID) <- M is N+5.
lp5([],[]).
lp5([N|L],[M|K]) :-
    p5(N,M,L),
    lp5(L,K).

query(lp5([1,1],_)).
```

identifier distinguishes calls

Stochastic Memoization

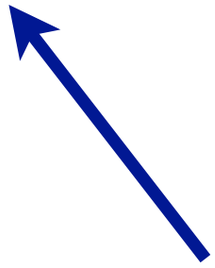
```
(define randplus5 (mem (lambda (x) (if (flip 0.6) (+ x 5) x))))

(map randplus5 '(1 1))
```



remember first value &
reuse for all later calls

Stochastic Memoization

```
(define randplus5 (mem (lambda (x) (if (flip 0.6) (+ x 5) x))))

(map randplus5 '(1 1))
```



remember first value &
reuse for all later calls

ProbLog always memoizes
PRISM never memoizes
Church allows fine-grained choice

ProbLog	**PRISM**	**Church**
probabilistic facts & choices	probabilistic choices	random primitives
all RVs memoized	no RVs memoized	user-defined per RV
Prolog	Prolog with mutually exclusive derivations	λ-calculus functions
distribution over worlds	distribution over derivations / answers	distribution over computations / answers

Roadmap

- Modeling (with detours to related work)
- Reasoning (and a bit of learning)
- Language extensions

Reasoning

- Exact inference with knowledge compilation
  - using proofs
  - using models
  - in PRISM
- Approximate inference

Answering Questions

Given:

program

queries

evidence

?

Find:

marginal probabilities

conditional probabilities

MPE state

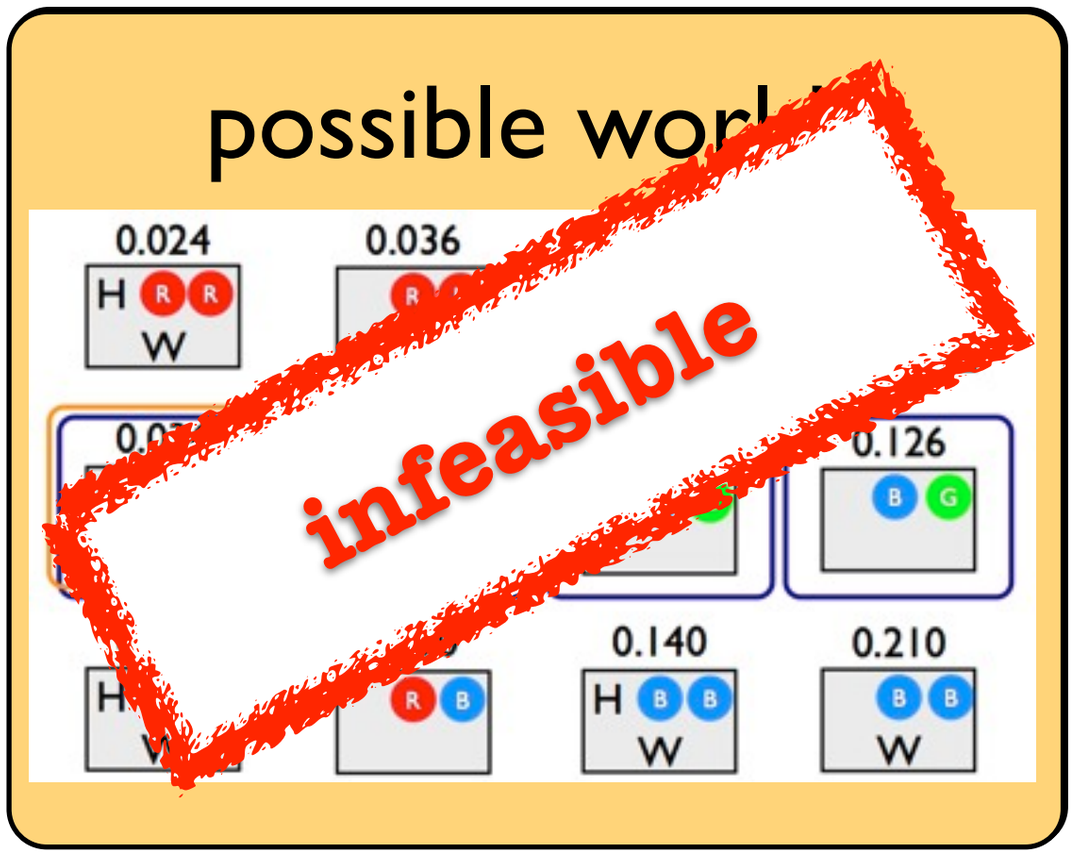
Answering Questions

Given:

program

queries

evidence



Find:

marginal probabilities

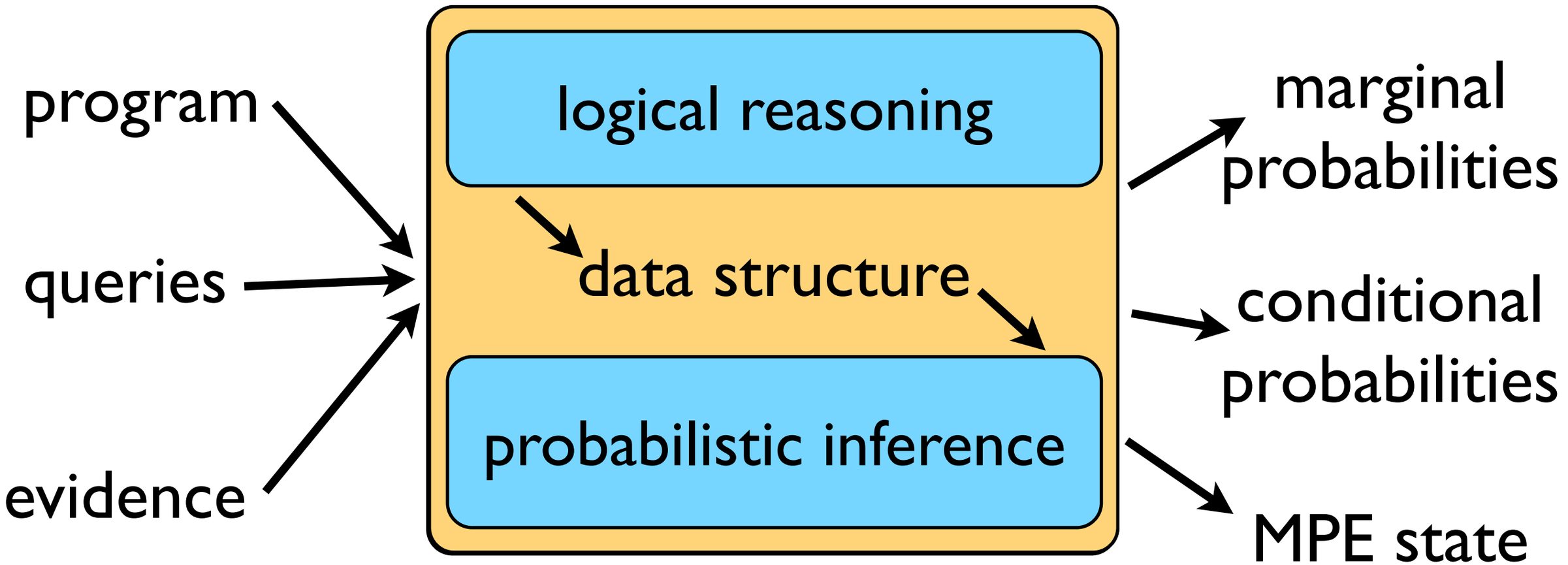
conditional probabilities

MPE state

Answering Questions

Given:

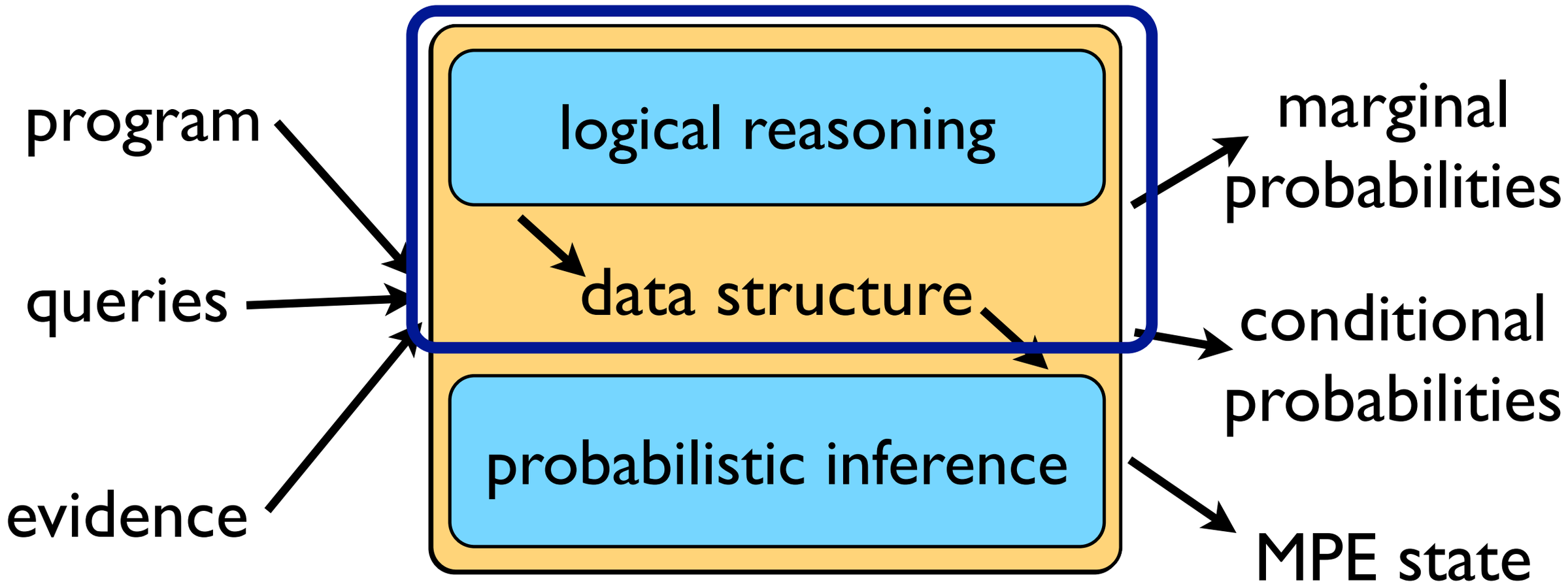
Find:



Answering Questions

Given:

Find:

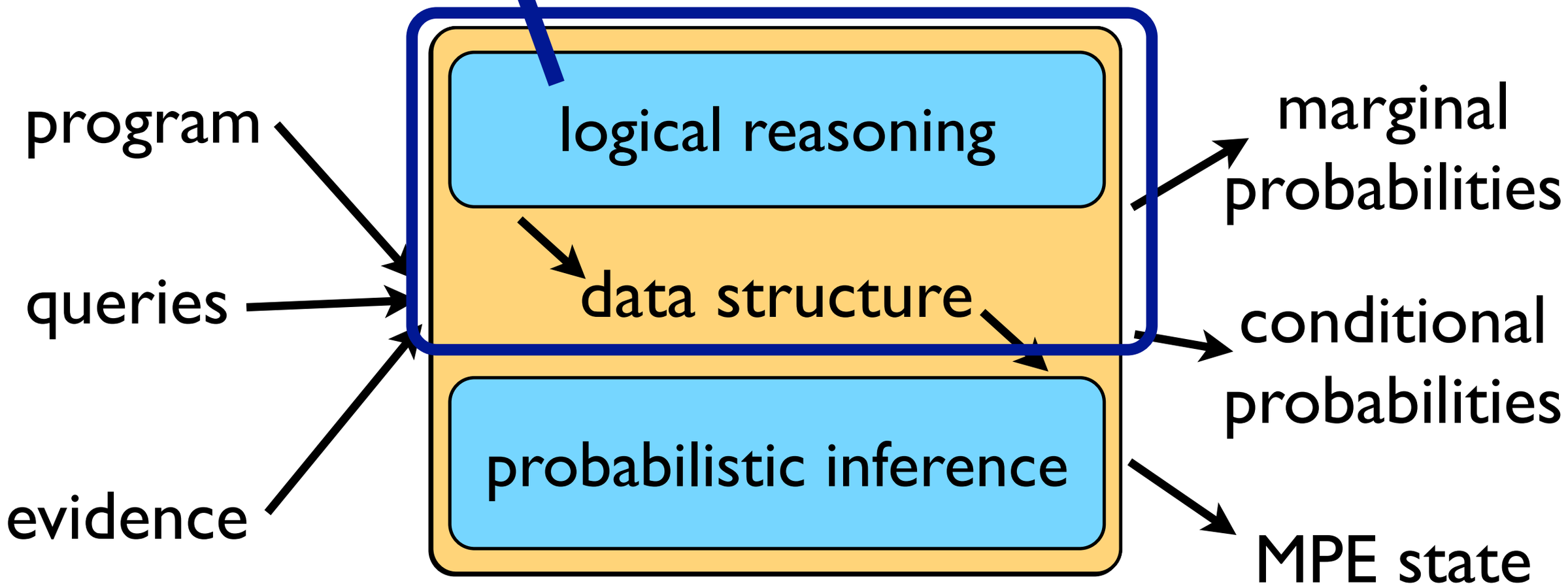


Answering Questions

1. using proofs
2. using models

knowledge compilation

Given:



Find:

Logical Reasoning: Proofs in Prolog

```
stress(ann).
influences(ann,bob).
influences(bob,carl).

smokes(X) :- stress(X).
smokes(X) :-
      influences(Y,X),
      smokes(Y).
```

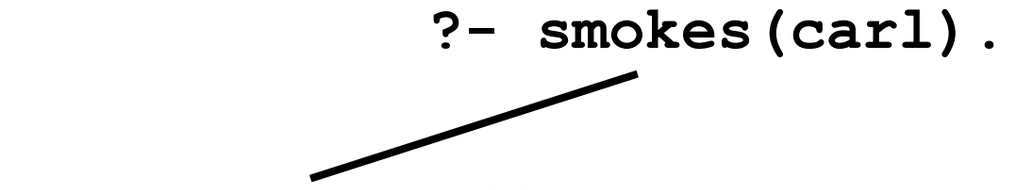
Logical Reasoning: Proofs in Prolog

```
stress(ann).
influences(ann,bob).
influences(bob,carl).

smokes(X) :- stress(X).
smokes(X) :-
      influences(Y,X),
      smokes(Y).
```

```
?- smokes(carl).
```

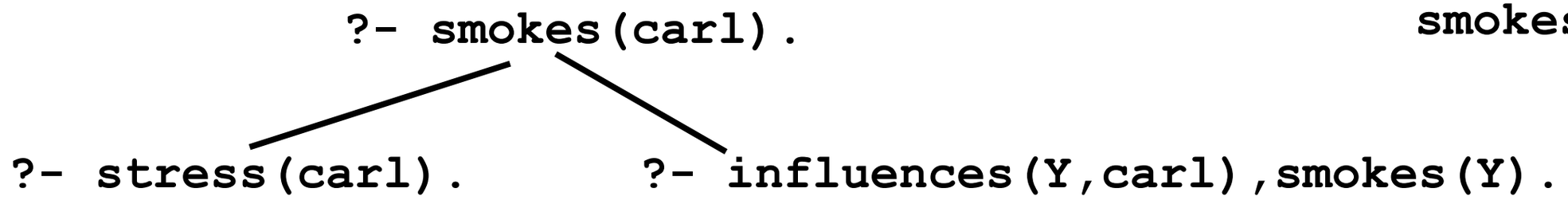
Logical Reasoning: Proofs in Prolog

```
stress(ann).
influences(ann,bob).
influences(bob,carl).

smokes(X) :- stress(X).
smokes(X) :-
      influences(Y,X),
      smokes(Y).
```



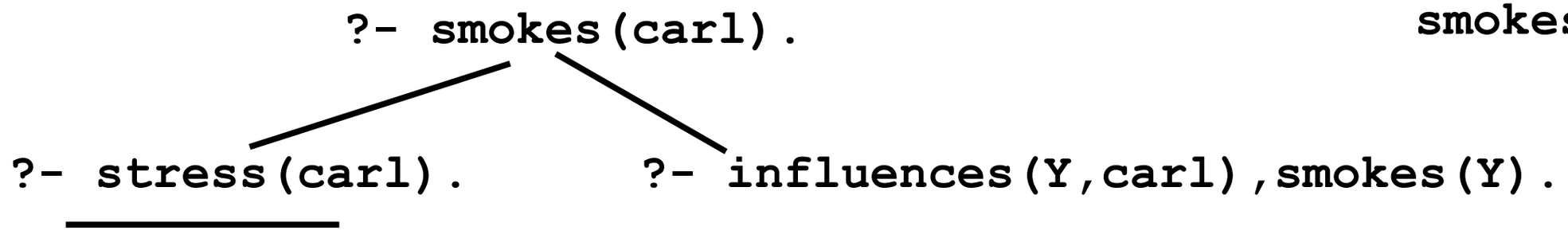
Logical Reasoning: Proofs in Prolog

```
stress(ann).
influences(ann,bob).
influences(bob,carl).

smokes(X)  :- stress(X).
smokes(X)  :-
      influences(Y,X),
      smokes(Y).
```



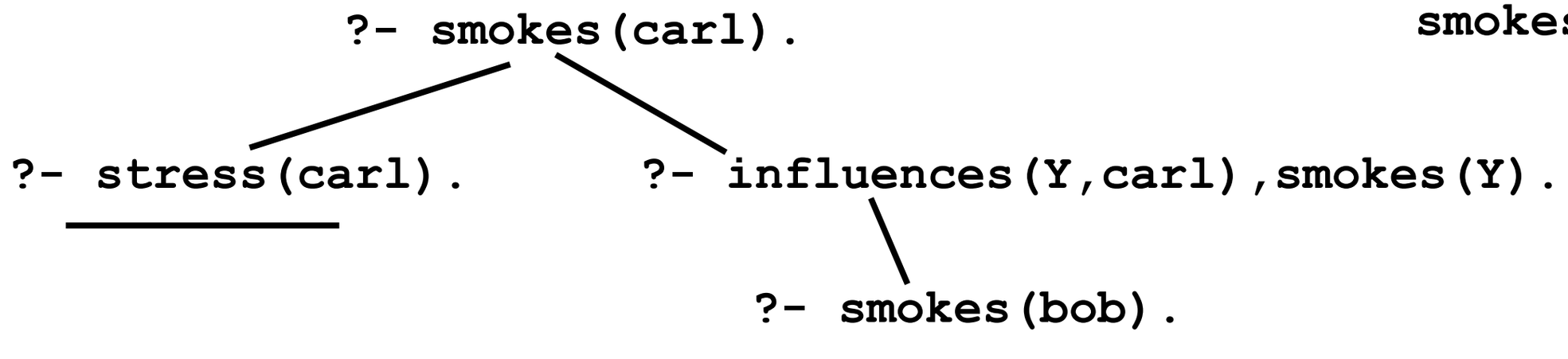
Logical Reasoning: Proofs in Prolog

```
stress(ann).
influences(ann,bob).
influences(bob,carl).

smokes(X)  :- stress(X).
smokes(X)  :-
      influences(Y,X),
      smokes(Y).
```



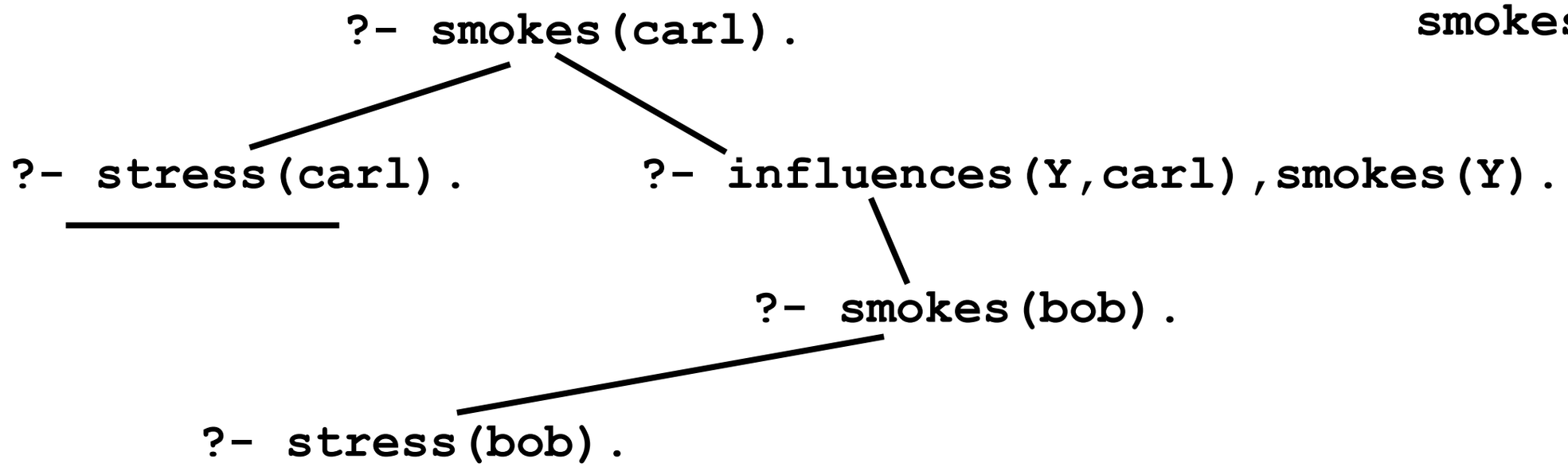
Logical Reasoning: Proofs in Prolog

```
stress(ann).
influences(ann,bob).
influences(bob,carl).

smokes(X)  :- stress(X).
smokes(X)  :-
      influences(Y,X),
      smokes(Y).
```



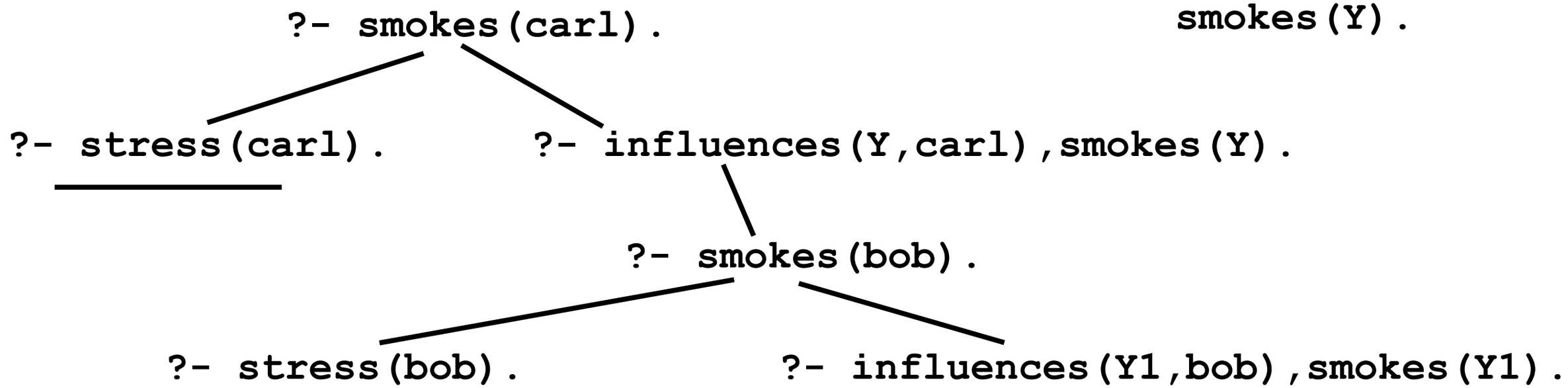
Logical Reasoning: Proofs in Prolog

```
stress(ann).
influences(ann,bob).
influences(bob,carl).

smokes(X)  :- stress(X).
smokes(X)  :-
      influences(Y,X),
      smokes(Y).
```



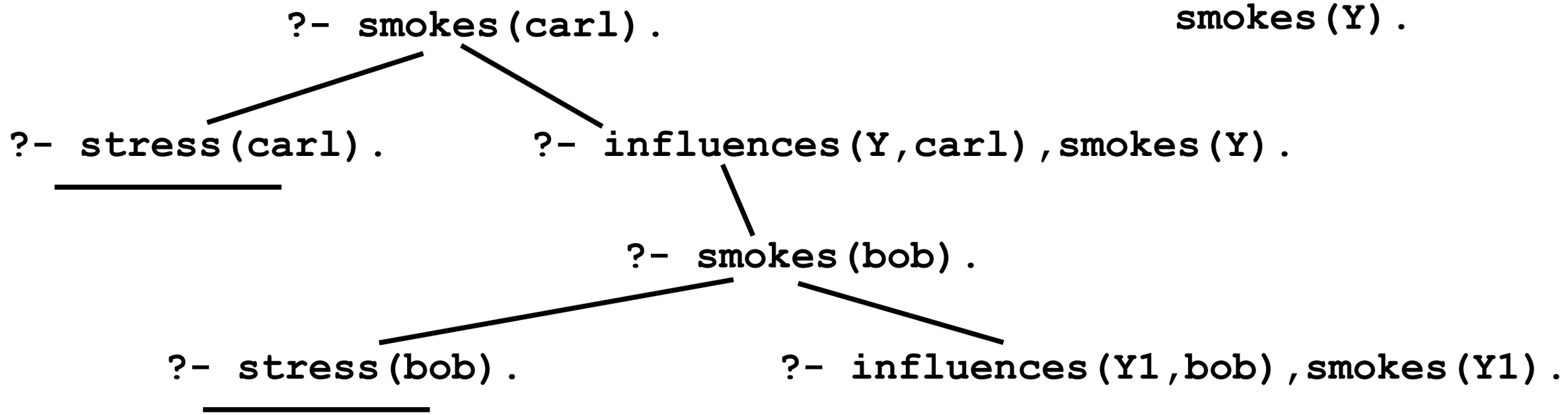
Logical Reasoning: Proofs in Prolog

```
stress(ann).
influences(ann,bob).
influences(bob,carl).

smokes(X)  :- stress(X).
smokes(X)  :-
      influences(Y,X),
      smokes(Y).
```



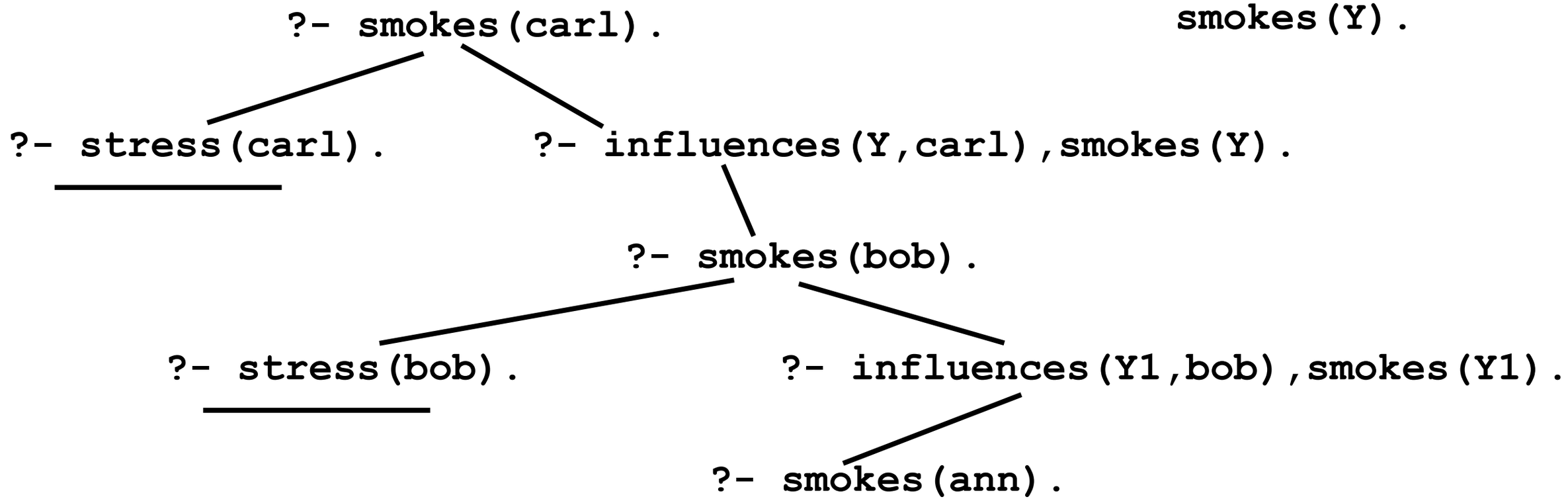
Logical Reasoning: Proofs in Prolog

```
stress(ann).
influences(ann,bob).
influences(bob,carl).

smokes(X)  :- stress(X).
smokes(X)  :-
      influences(Y,X),
      smokes(Y).
```



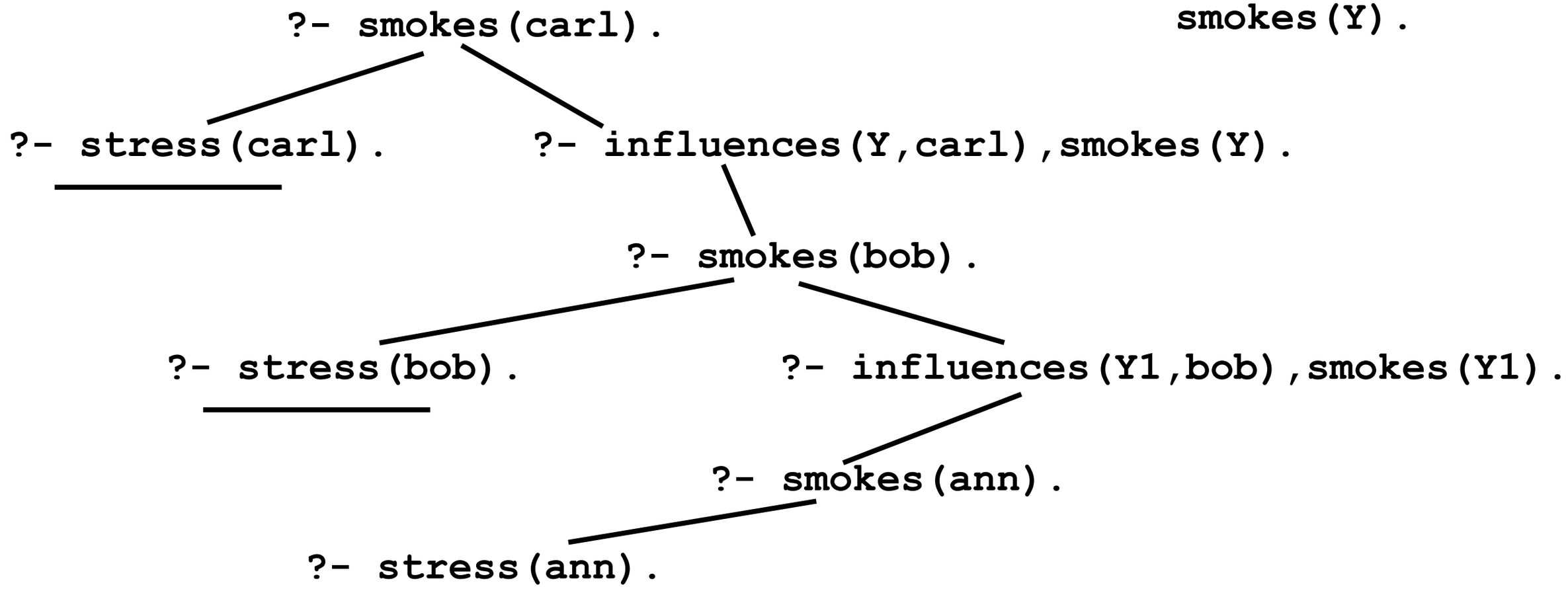
Logical Reasoning: Proofs in Prolog

```
stress(ann).
influences(ann,bob).
influences(bob,carl).

smokes(X)  :- stress(X).
smokes(X)  :-
      influences(Y,X),
      smokes(Y).
```



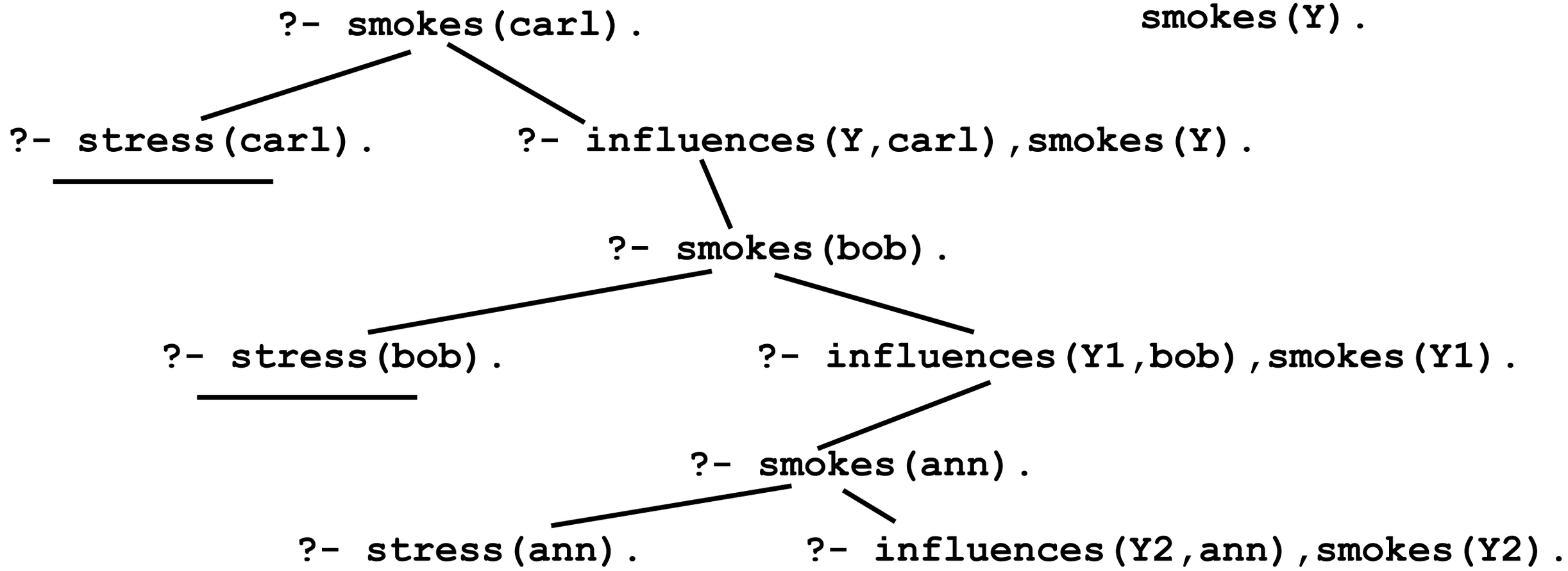
Logical Reasoning: Proofs in Prolog

```
stress(ann).
influences(ann,bob).
influences(bob,carl).

smokes(X)  :- stress(X).
smokes(X)  :-
      influences(Y,X),
      smokes(Y).
```



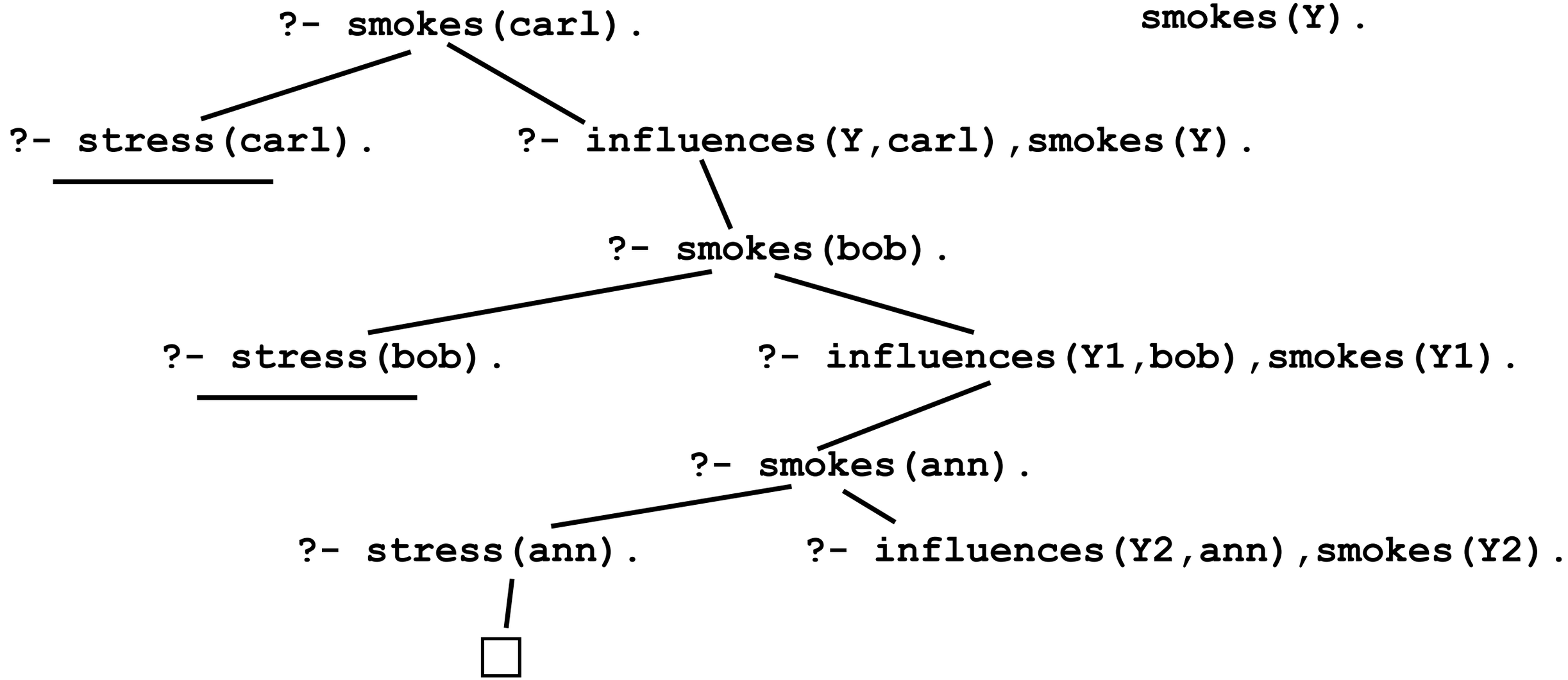
Logical Reasoning: Proofs in Prolog

```
stress(ann).
influences(ann,bob).
influences(bob,carl).

smokes(X)  :- stress(X).
smokes(X)  :-
      influences(Y,X),
      smokes(Y).
```



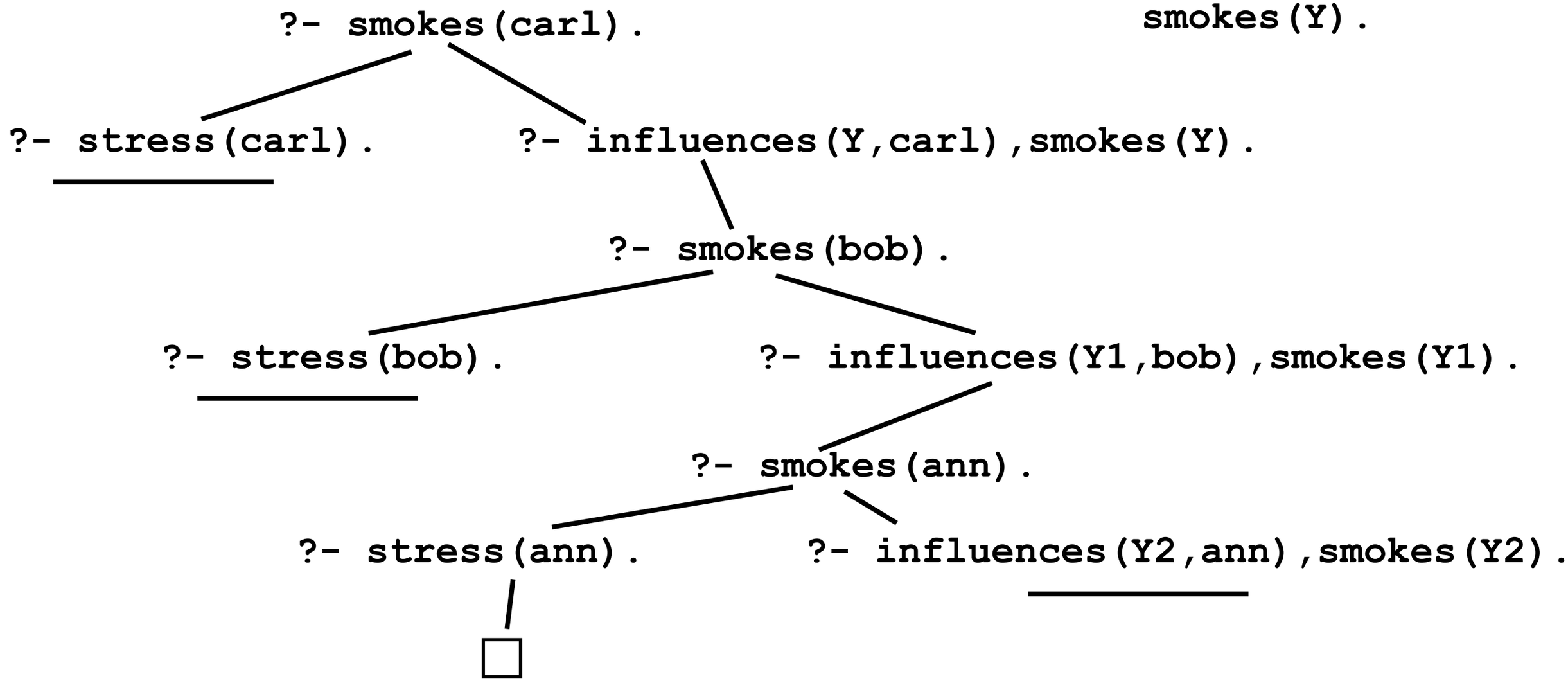
Logical Reasoning: Proofs in Prolog

```
stress(ann).
influences(ann,bob).
influences(bob,carl).

smokes(X)  :- stress(X).
smokes(X)  :-
      influences(Y,X),
      smokes(Y).
```



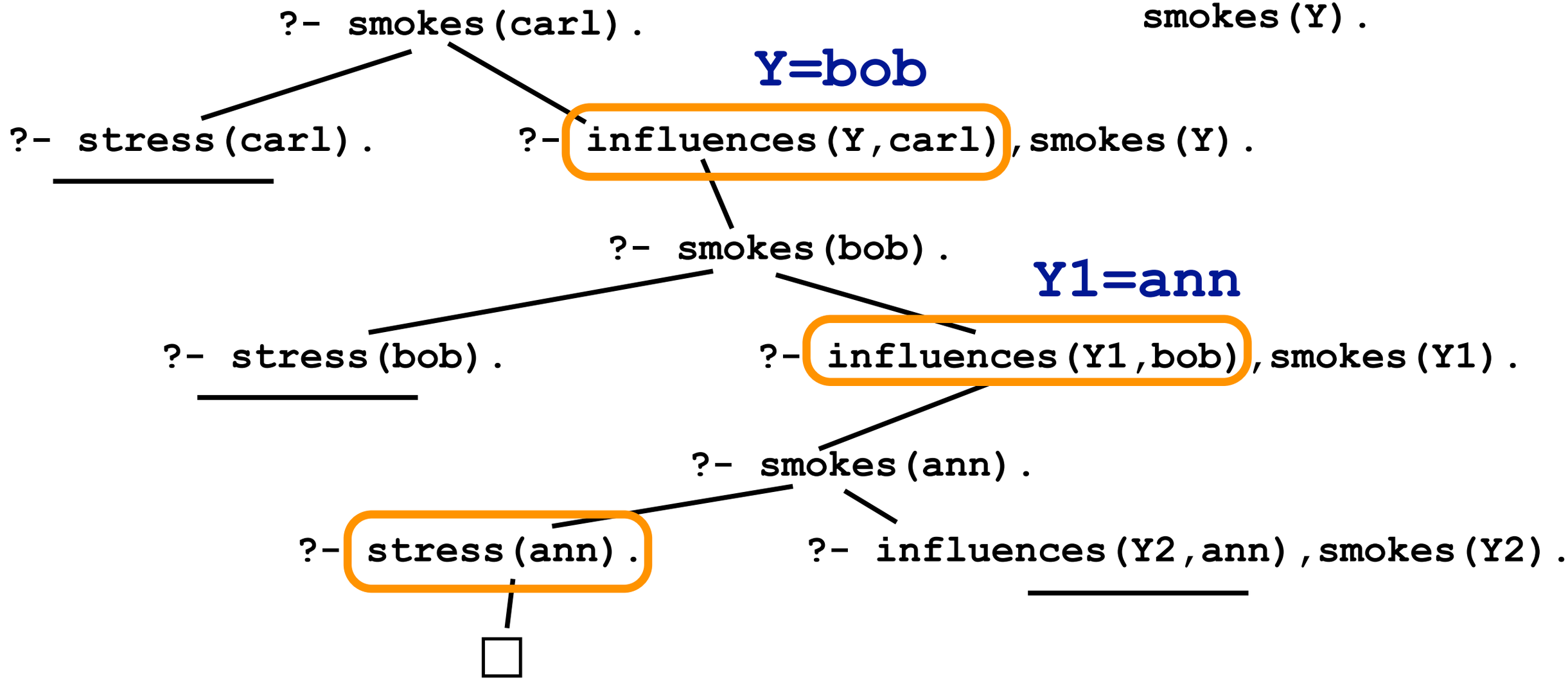
Logical Reasoning: Proofs in Prolog

```
stress(ann).
influences(ann,bob).
influences(bob,carl).

smokes(X)  :- stress(X).
smokes(X)  :-
      influences(Y,X),
      smokes(Y).
```



Logical Reasoning: Proofs in Prolog

```
stress(ann).
influences(ann,bob).
influences(bob,carl).

smokes(X) :- stress(X).
smokes(X) :-
      influences(Y,X),
      smokes(Y).
```

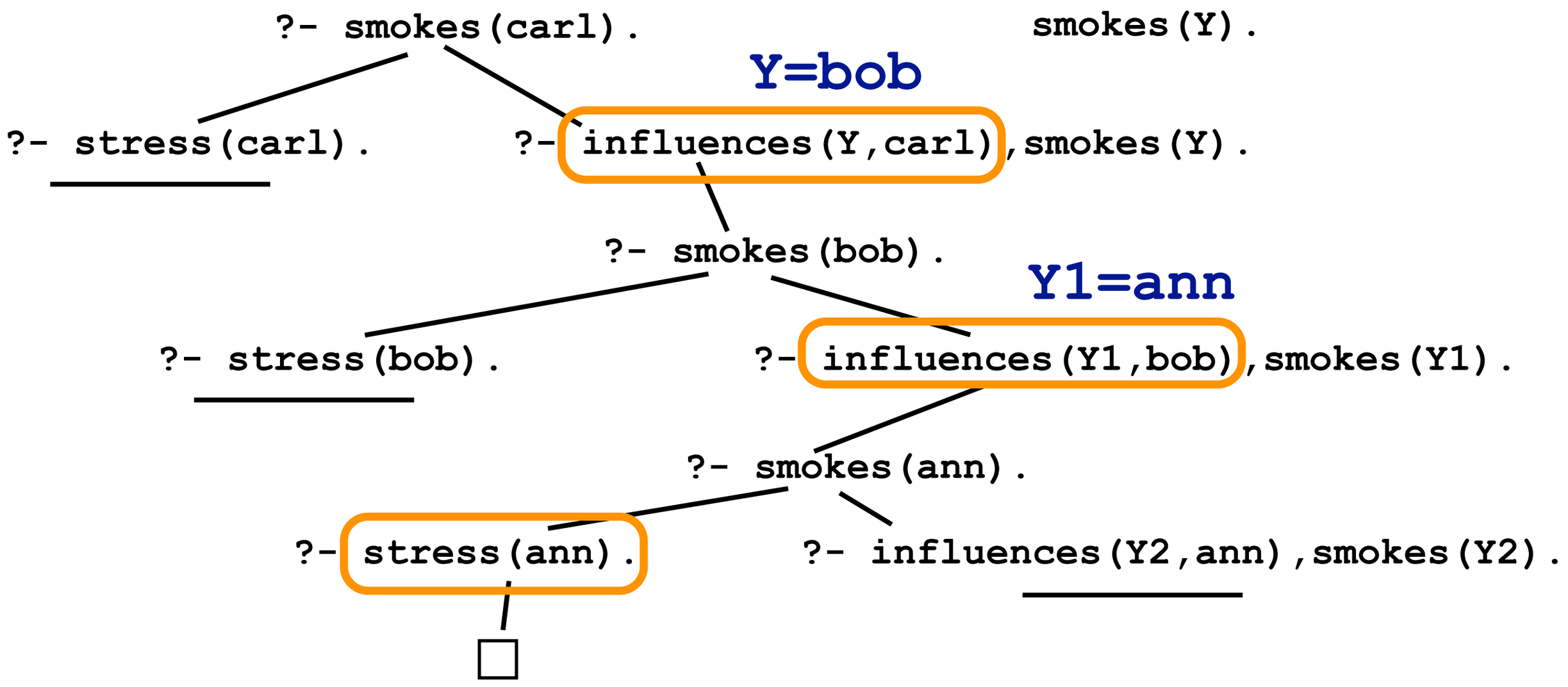


proof = facts used in successful derivation:

`influences(bob,carl) &influences(ann,bob) &stress(ann)`

Proofs in ProbLog

```
0.8::stress(ann).
0.6::influences(ann,bob).
0.2::influences(bob,carl).

smokes(X) :- stress(X).
smokes(X) :-
     influences(Y,X),
     smokes(Y).
```

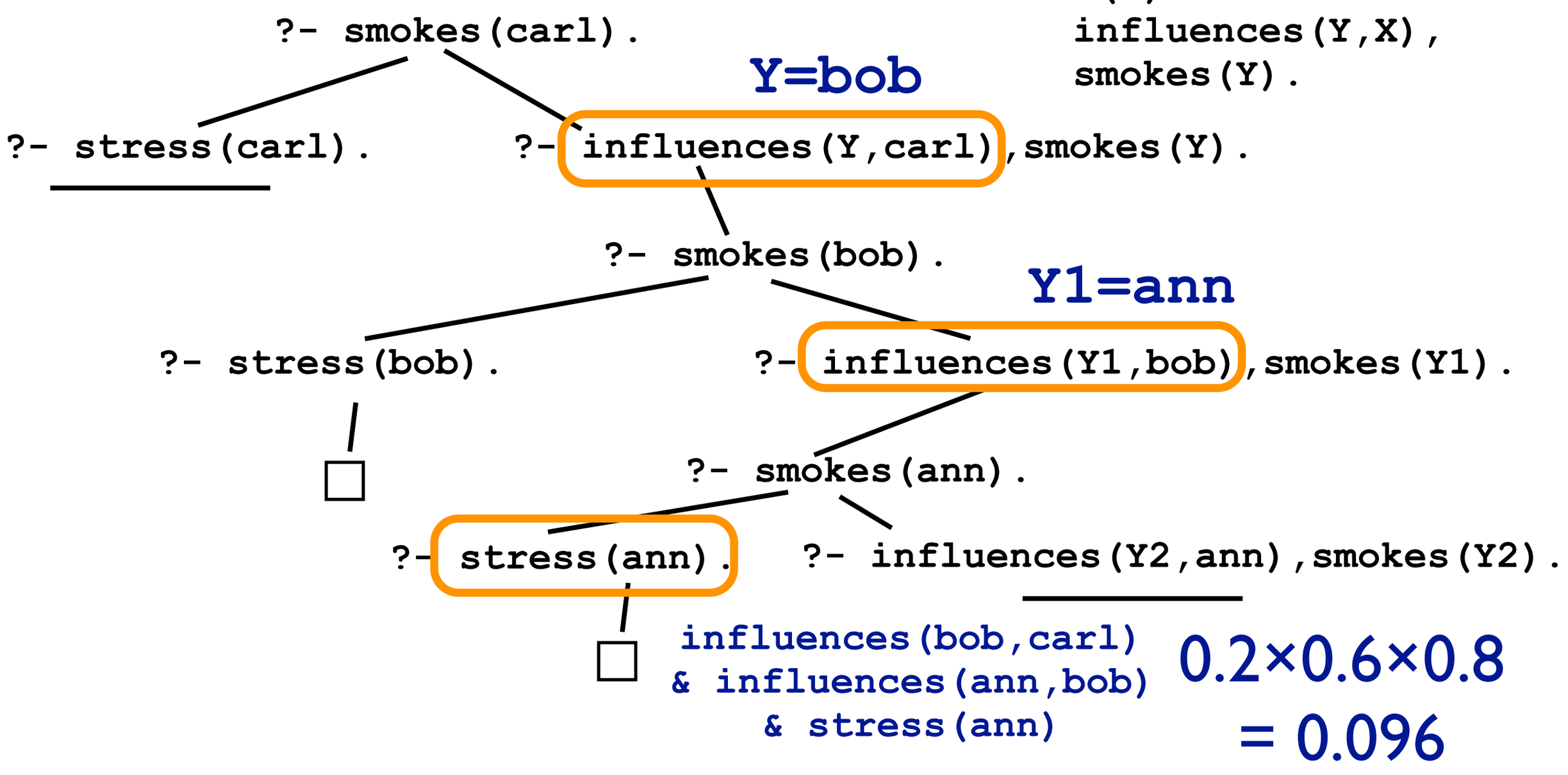


`influences(bob,carl)&influences(ann,bob)&stress(ann)`

probability of proof = 0.2 × 0.6 × 0.8 = 0.096

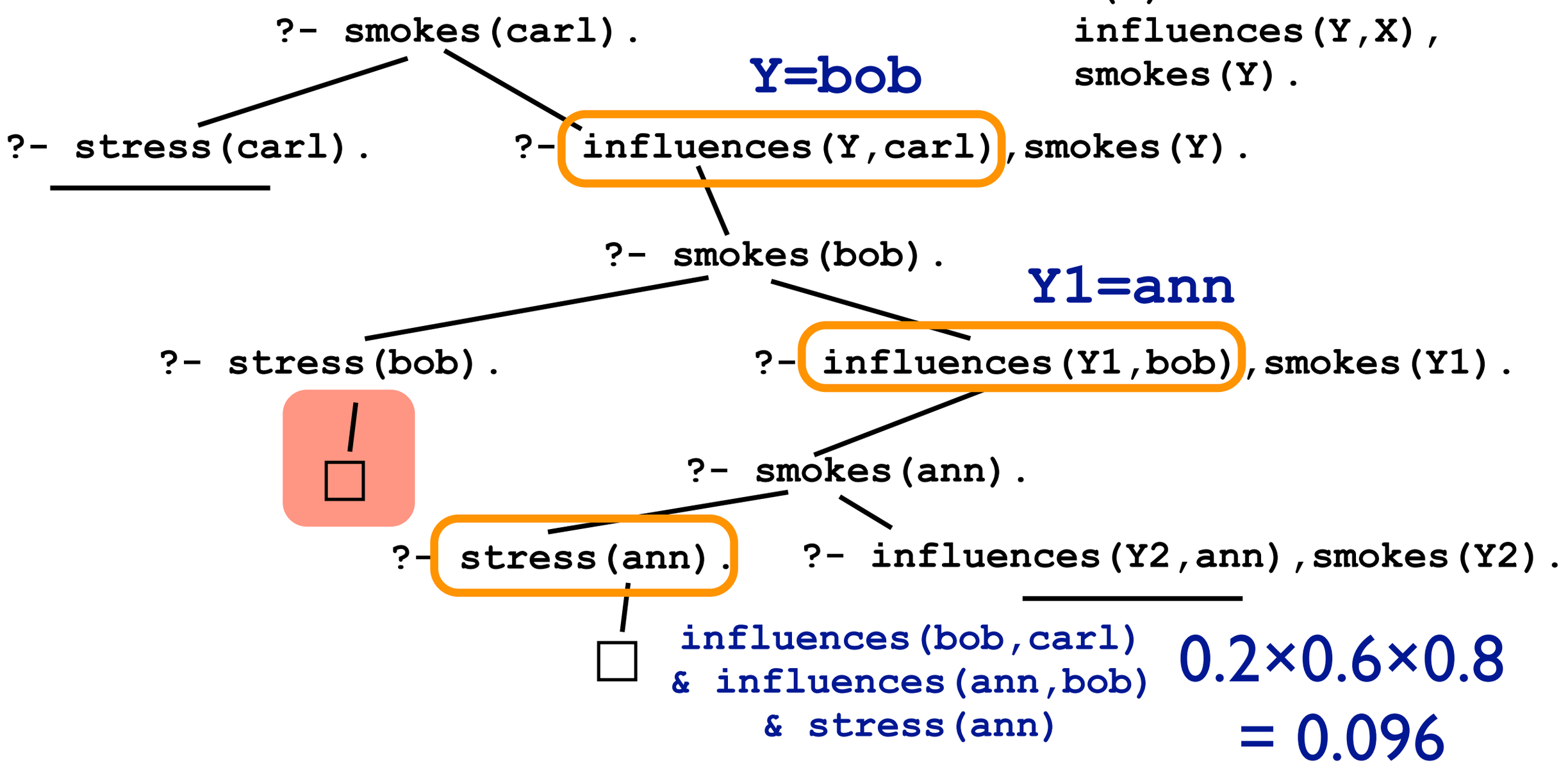
Proofs in ProbLog

```
0.8::stress(ann).
0.4::stress(bob).
0.6::influences(ann,bob).
0.2::influences(bob,carl).

smokes(X) :- stress(X).
smokes(X) :-
     influences(Y,X),
     smokes(Y).
```



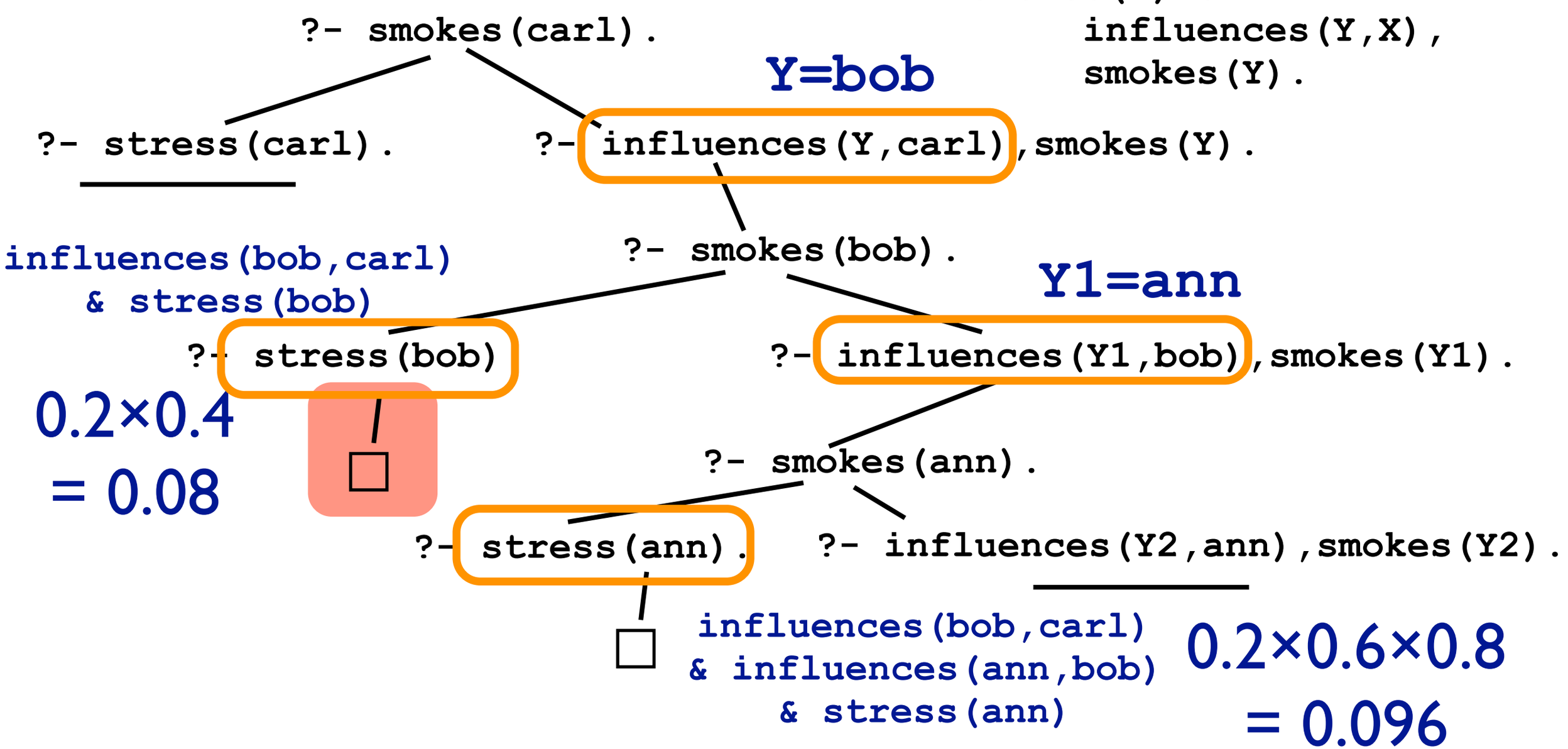
Proofs in ProbLog

```
0.8::stress(ann).
0.4::stress(bob).
0.6::influences(ann,bob).
0.2::influences(bob,carl).

smokes(X) :- stress(X).
smokes(X) :-
     influences(Y,X),
     smokes(Y).
```



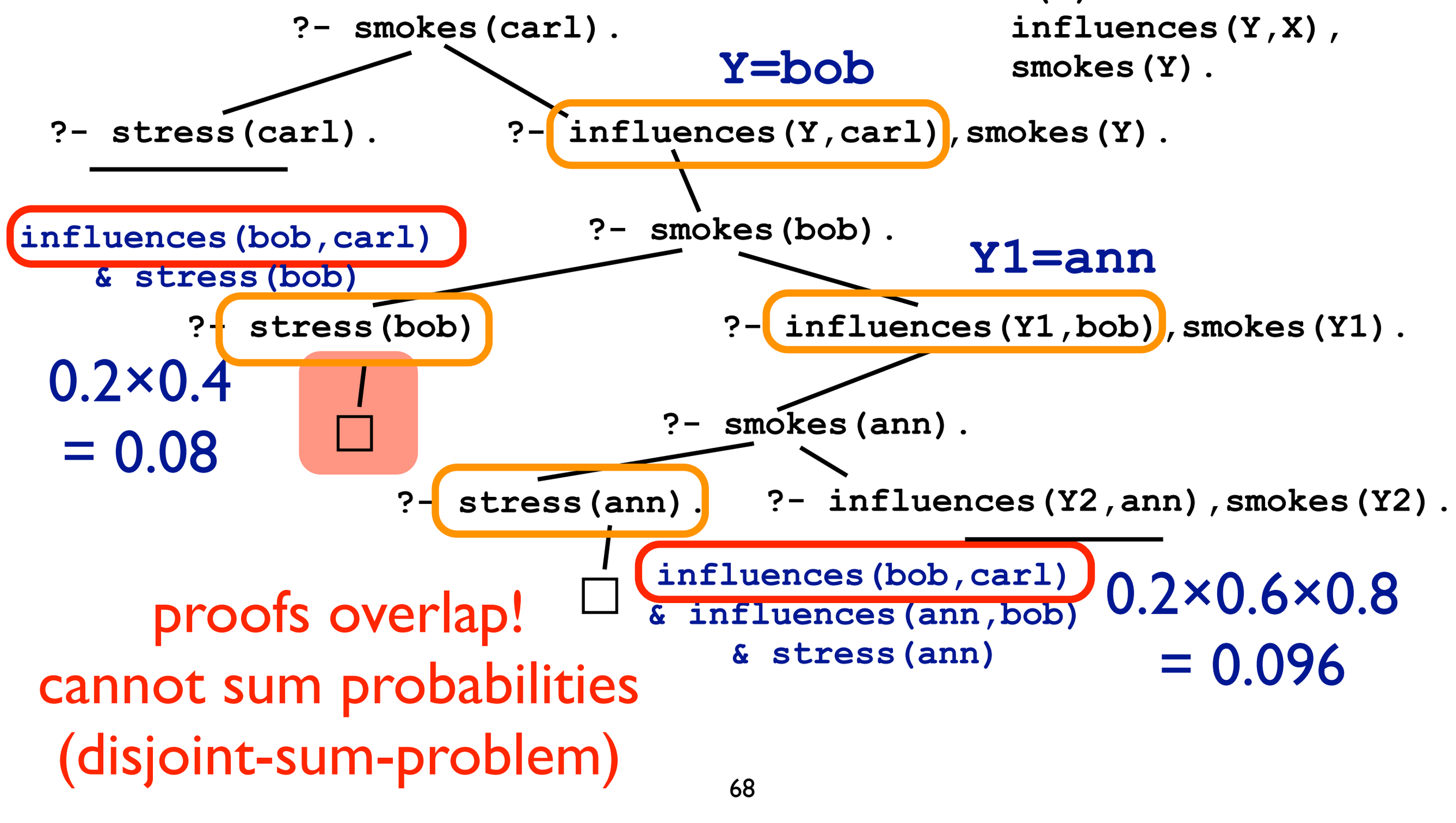
Proofs in ProbLog

```
0.8::stress(ann).
0.4::stress(bob).
0.6::influences(ann,bob).
0.2::influences(bob,carl).

smokes(X) :- stress(X).
smokes(X) :-
     influences(Y,X),
     smokes(Y).
```



Proofs in ProbLog

```
0.8::stress(ann).
0.4::stress(bob).
0.6::influences(ann,bob).
0.2::influences(bob,carl).

smokes(X) :- stress(X).
smokes(X) :-
     influences(Y,X),
     smokes(Y).
```



proofs overlap!
cannot sum probabilities
(disjoint-sum-problem)

Disjoint-Sum-Problem

possible worlds

```
infl(bob,carl) &   infl(ann,bob) &   st(ann) & \+st(bob)
infl(bob,carl) &   infl(ann,bob) &   st(ann) &   st(bob)
infl(bob,carl) & \+infl(ann,bob) &   st(ann) &   st(bob)
infl(bob,carl) &   infl(ann,bob) & \+st(ann) &   st(bob)
infl(bob,carl) & \+infl(ann,bob) & \+st(ann) &   st(bob)
...
```

Disjoint-Sum-Problem

possible worlds

influences(bob,carl) &
influences(ann,bob) & stress(ann)

```
infl(bob,carl) &   infl(ann,bob) &   st(ann) & \+st(bob)

infl(bob,carl) &   infl(ann,bob) &   st(ann) &   st(bob)

infl(bob,carl) & \+infl(ann,bob) &   st(ann) &   st(bob)

infl(bob,carl) &   infl(ann,bob) & \+st(ann) &   st(bob)

infl(bob,carl) & \+infl(ann,bob) & \+st(ann) &   st(bob)

...
```

Disjoint-Sum-Problem

possible worlds

influences(bob,carl) &
influences(ann,bob) & stress(ann)

```
infl(bob,carl) &   infl(ann,bob) &   st(ann) & \+st(bob)
infl(bob,carl) &   infl(ann,bob) &   st(ann) &   st(bob)
infl(bob,carl) & \+infl(ann,bob) &   st(ann) &   st(bob)
infl(bob,carl) &   infl(ann,bob) & \+st(ann) &   st(bob)
infl(bob,carl) & \+infl(ann,bob) & \+st(ann) &   st(bob)
```

... **influences(bob,carl) & stress(bob)**

Disjoint-Sum-Problem

possible worlds

influences(bob,carl) &
influences(ann,bob) & stress(ann)

```
infl(bob,carl) &   infl(ann,bob) &   st(ann) & \+st(bob)
infl(bob,carl) &   infl(ann,bob) &   st(ann) &   st(bob)
infl(bob,carl) & \+infl(ann,bob) &   st(ann) &   st(bob)
infl(bob,carl) &   infl(ann,bob) & \+st(ann) &   st(bob)
infl(bob,carl) & \+infl(ann,bob) & \+st(ann) &   st(bob)
```

... **influences(bob,carl) & stress(bob)**

sum of proof probabilities: 0.096+0.08 = 0.1760

Disjoint-Sum-Problem

possible worlds

influences(bob,carl) &
influences(ann,bob) & stress(ann)

infl(bob,carl) & infl(ann,bob) & st(ann) & \+st(bob)	0.0576
infl(bob,carl) & infl(ann,bob) & st(ann) & st(bob)	0.0384
infl(bob,carl) & \+infl(ann,bob) & st(ann) & st(bob)	0.0256
infl(bob,carl) & infl(ann,bob) & \+st(ann) & st(bob)	0.0096
infl(bob,carl) & \+infl(ann,bob) & \+st(ann) & st(bob)	0.0064
... **influences(bob,carl) & stress(bob)**	∑ = 0.1376

sum of proof probabilities: 0.096+0.08 = 0.1760

Disjoint-Sum-Problem

possible worlds

solution: knowledge compilation

`infl(bob,carl) & infl(ann,bob) & st(ann) & \+st(bob)`	0.0576
`infl(bob,carl) & infl(ann,bob) & st(ann) & st(bob)`	0.0384
`infl(bob,carl) & \+infl(ann,bob) & st(ann) & st(bob)`	0.0256
`infl(bob,carl) & infl(ann,bob) & \+st(ann) & st(bob)`	0.0096
`infl(bob,carl) & \+infl(ann,bob) & \+st(ann) & st(bob)`	0.0064
`...` `influences(bob,carl) & stress(bob)`	$\sum = 0.1376$

sum of proof probabilities: 0.096+0.08 = 0.1760

Binary Decision Diagrams [Bryant 86]

- compact graphical representation of Boolean formula
- popular in many branches of CS
- automatically disjoins proofs
  → efficient probability computation

Binary Decision Diagrams
[Bryant 86]

$X \vee Y \vee Z$

Binary Decision Diagrams
[Bryant 86]

$X \vee Y \vee Z$

X	0	0	0	0	1	1	1	1
Y	0	0	1	1	0	0	1	1
Z	0	1	0	1	0	1	0	1

Binary Decision Diagrams
[Bryant 86]

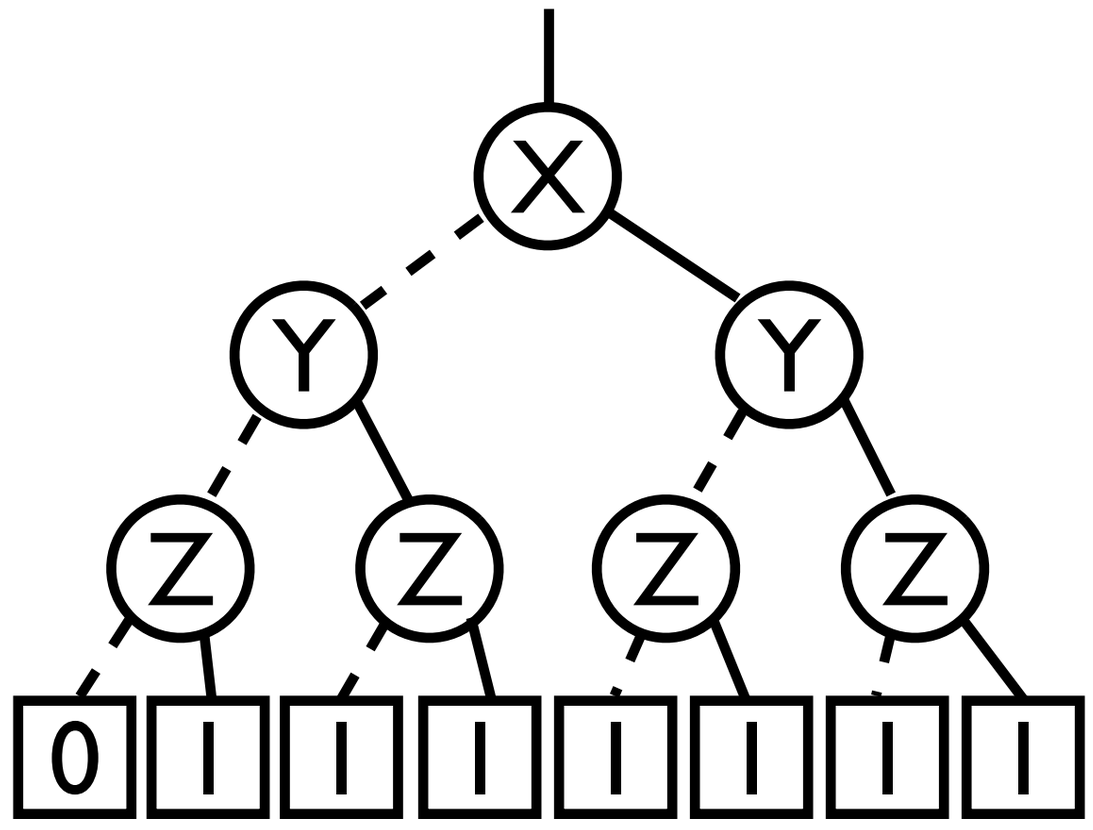
$X \vee Y \vee Z$

X	0	0	0	0	1	1	1	1
Y	0	0	1	1	0	0	1	1
Z	0	1	0	1	0	1	0	1

Binary Decision Diagrams

[Bryant 86]

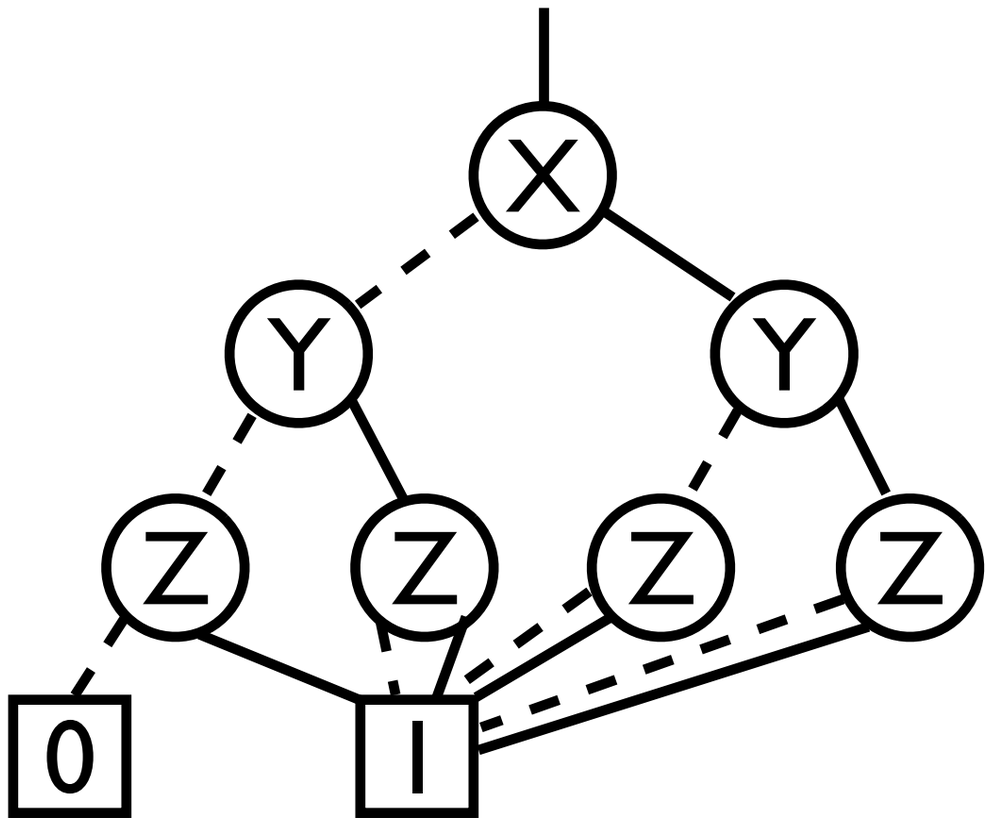
$X \vee Y \vee Z$



X	0	0	0	0	1	1	1	1
Y	0	0	1	1	0	0	1	1
Z	0	1	0	1	0	1	0	1

Binary Decision Diagrams

[Bryant 86]

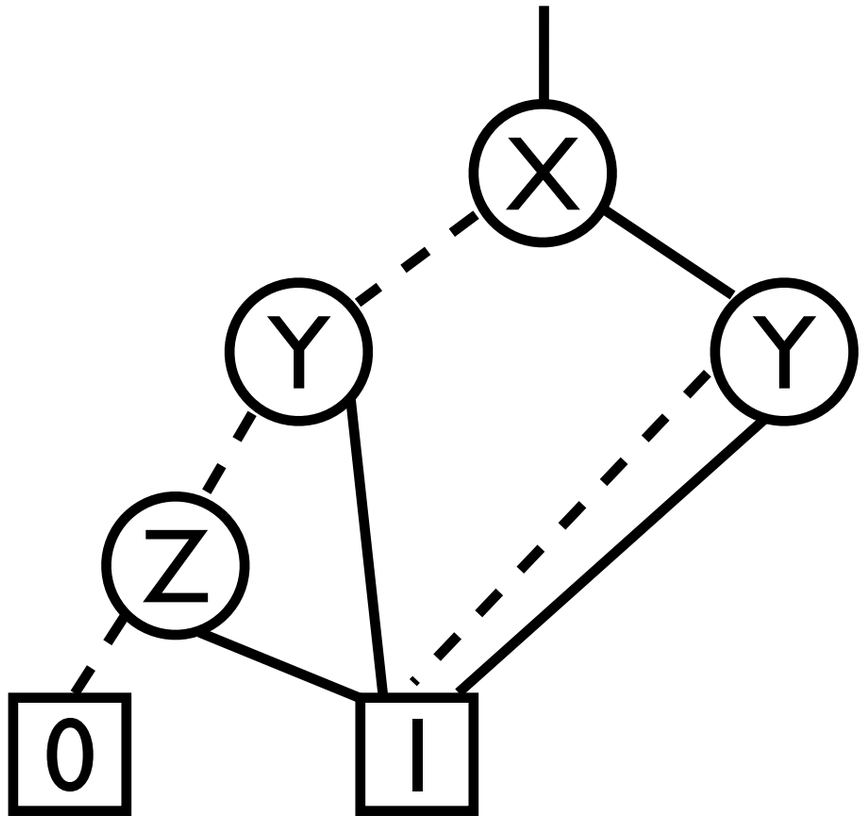


$X \vee Y \vee Z$

X	0	0	0	0	1	1	1	1
Y	0	0	1	1	0	0	1	1
Z	0	1	0	1	0	1	0	1

Binary Decision Diagrams

[Bryant 86]

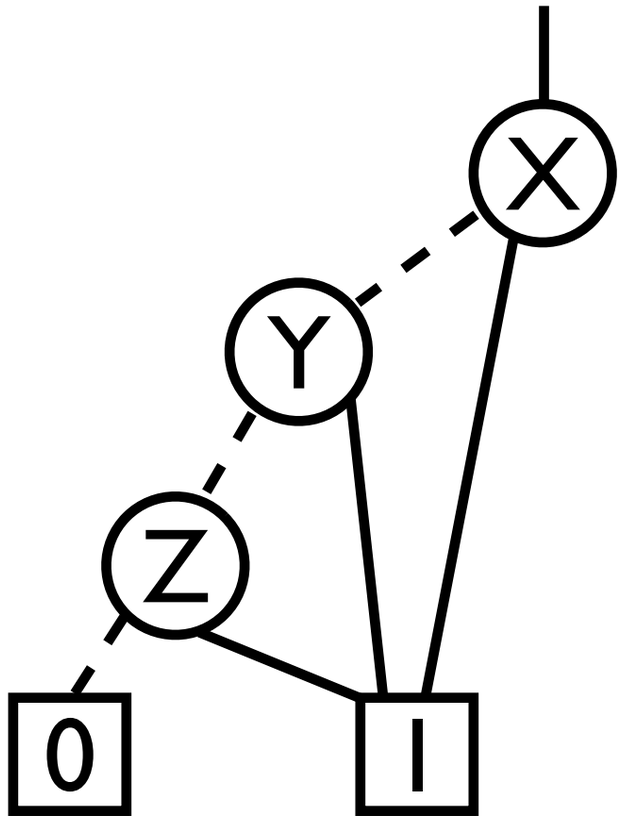


$X \vee Y \vee Z$

X	0	0	0	0	1	1	1	1
Y	0	0	1	1	0	0	1	1
Z	0	1	0	1	0	1	0	1

Binary Decision Diagrams

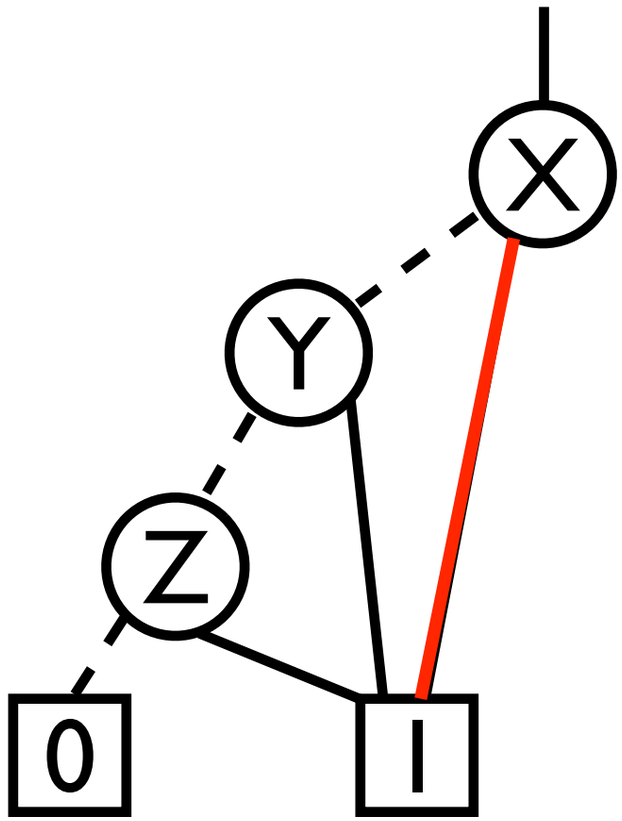
[Bryant 86]



$X \vee Y \vee Z$

X	0	0	0	0	1	1	1	1
Y	0	0	1	1	0	0	1	1
Z	0	1	0	1	0	1	0	1

Binary Decision Diagrams
[Bryant 86]

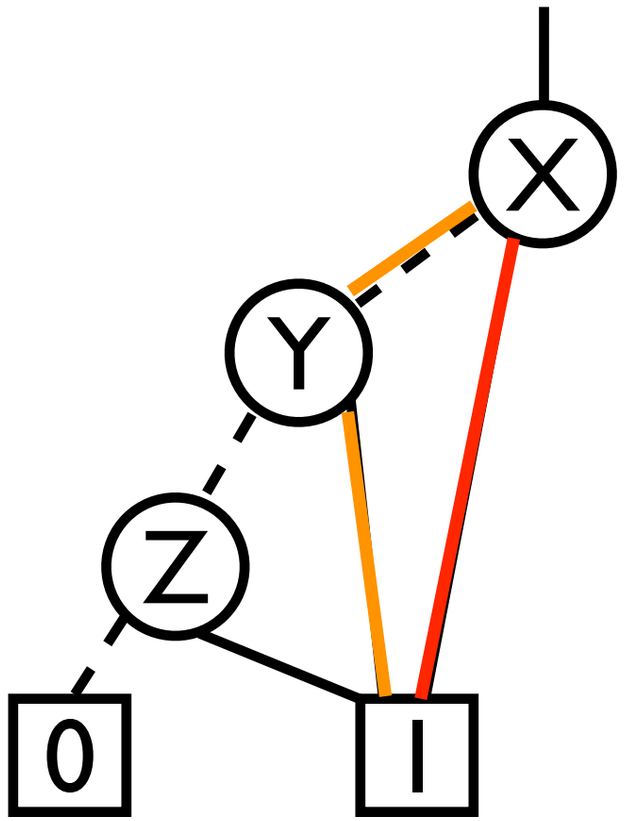


$X \vee Y \vee Z$

$X \vee (\neg X \wedge Y) \vee (\neg X \wedge \neg Y \wedge Z)$

X	0	0	0	0	1	1	1	1
Y	0	0	1	1	0	0	1	1
Z	0	1	0	1	0	1	0	1

Binary Decision Diagrams
[Bryant 86]

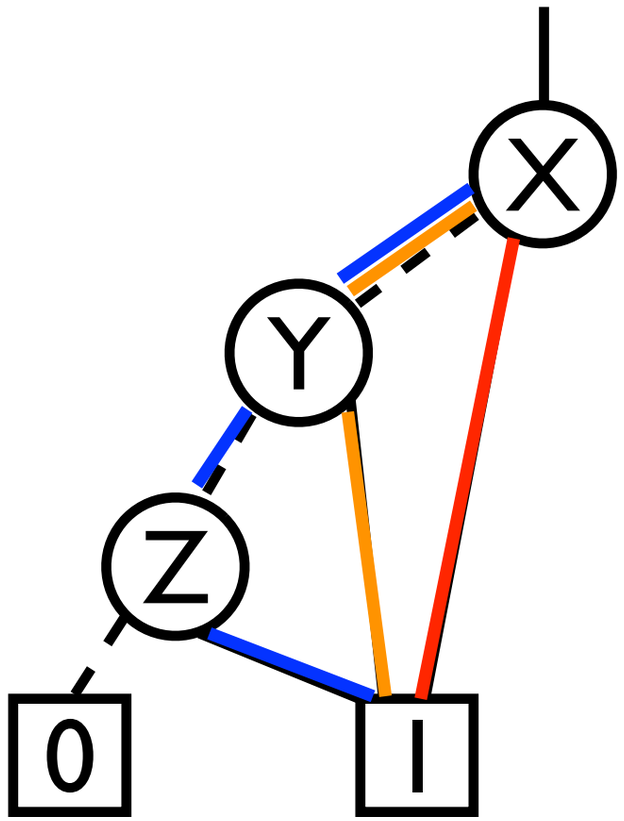


$X \vee Y \vee Z$

$X \vee (\neg X \wedge Y) \vee (\neg X \wedge \neg Y \wedge Z)$

X	0	0	0	0	1	1	1	1
Y	0	0	1	1	0	0	1	1
Z	0	1	0	1	0	1	0	1

Binary Decision Diagrams
[Bryant 86]

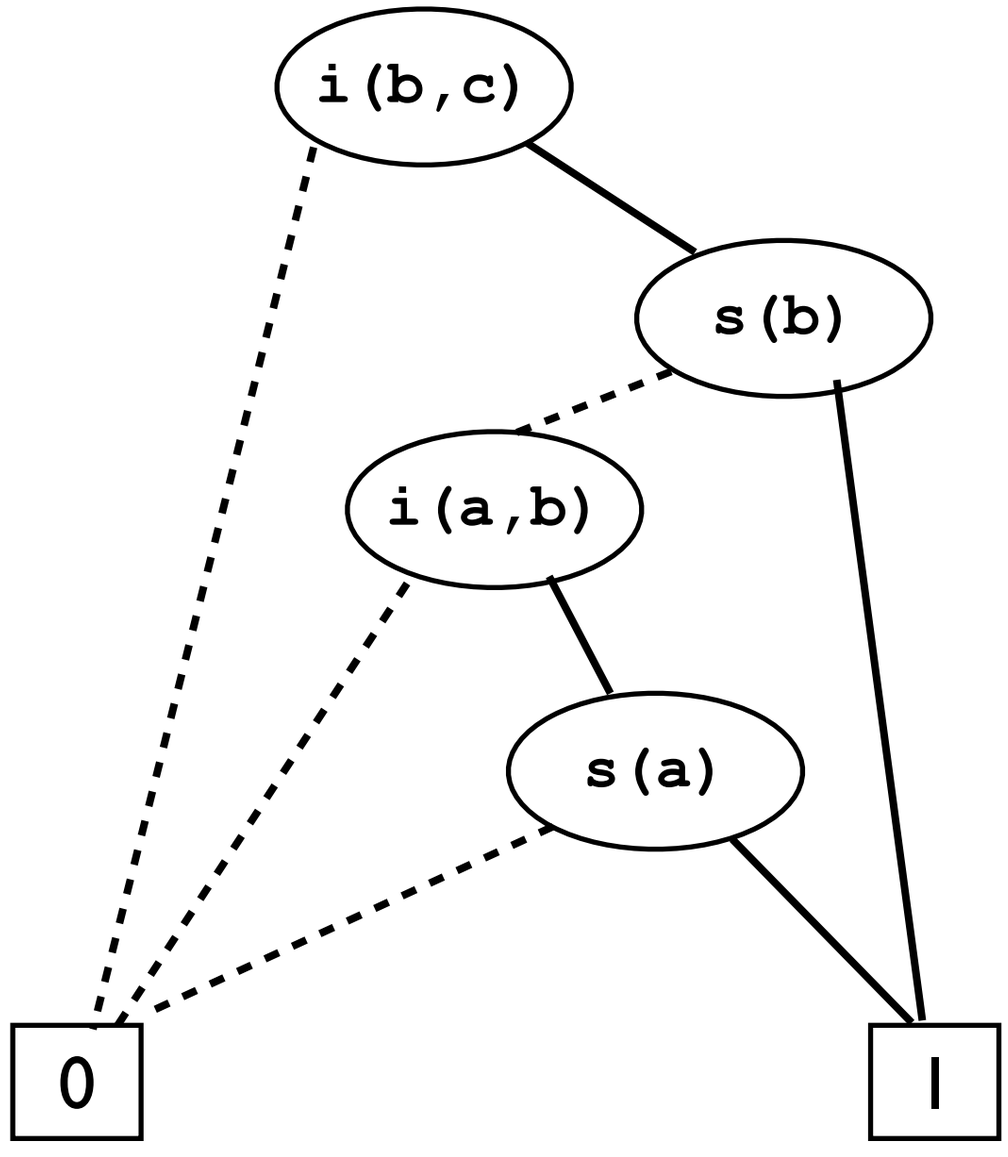


$$X \vee Y \vee Z$$

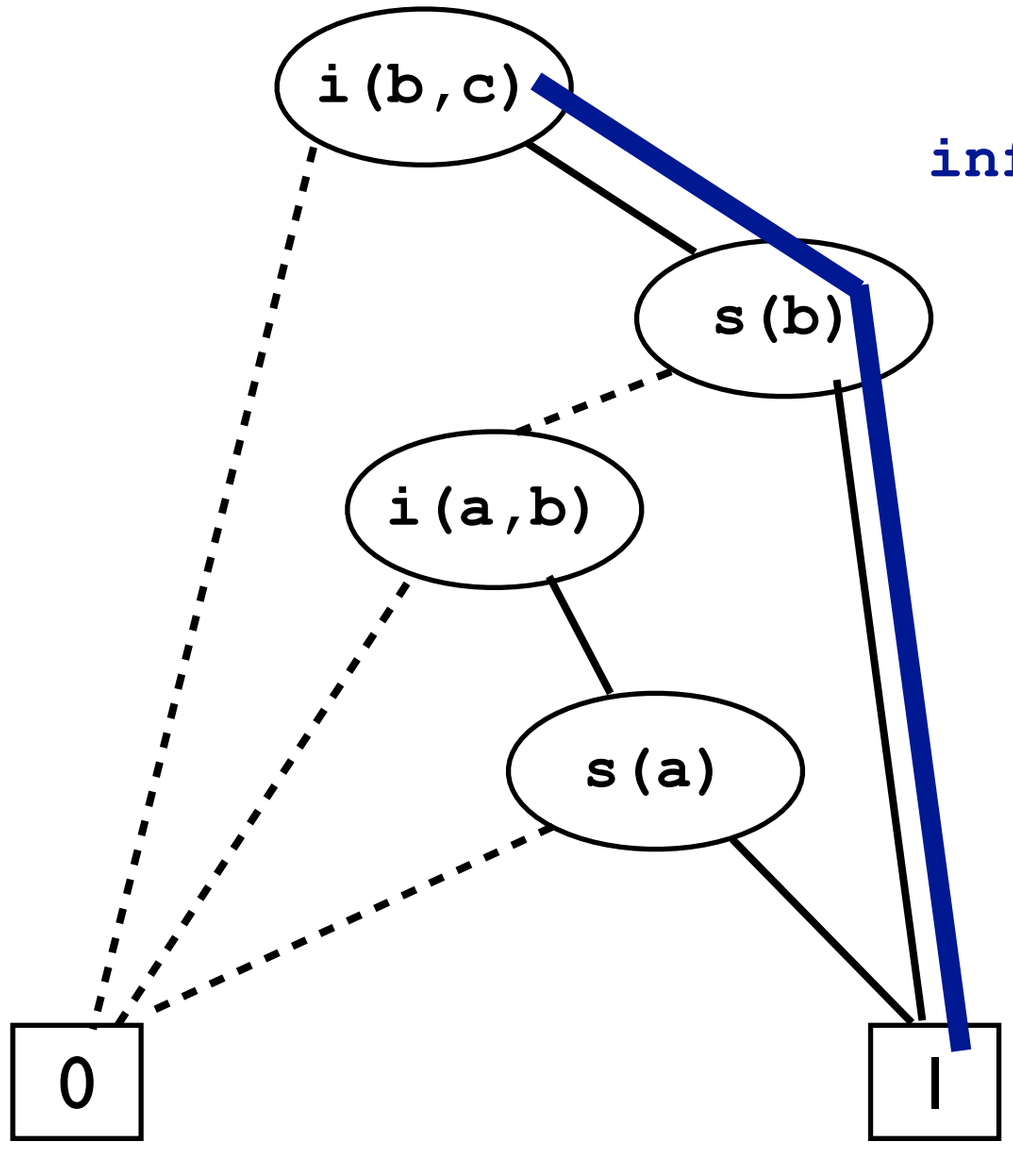
$$X \vee (\neg X \wedge Y) \vee (\neg X \wedge \neg Y \wedge Z)$$

X	0	0	0	0	1	1	1	1
Y	0	0	1	1	0	0	1	1
Z	0	1	0	1	0	1	0	1

Binary Decision Diagrams
[Bryant 86]



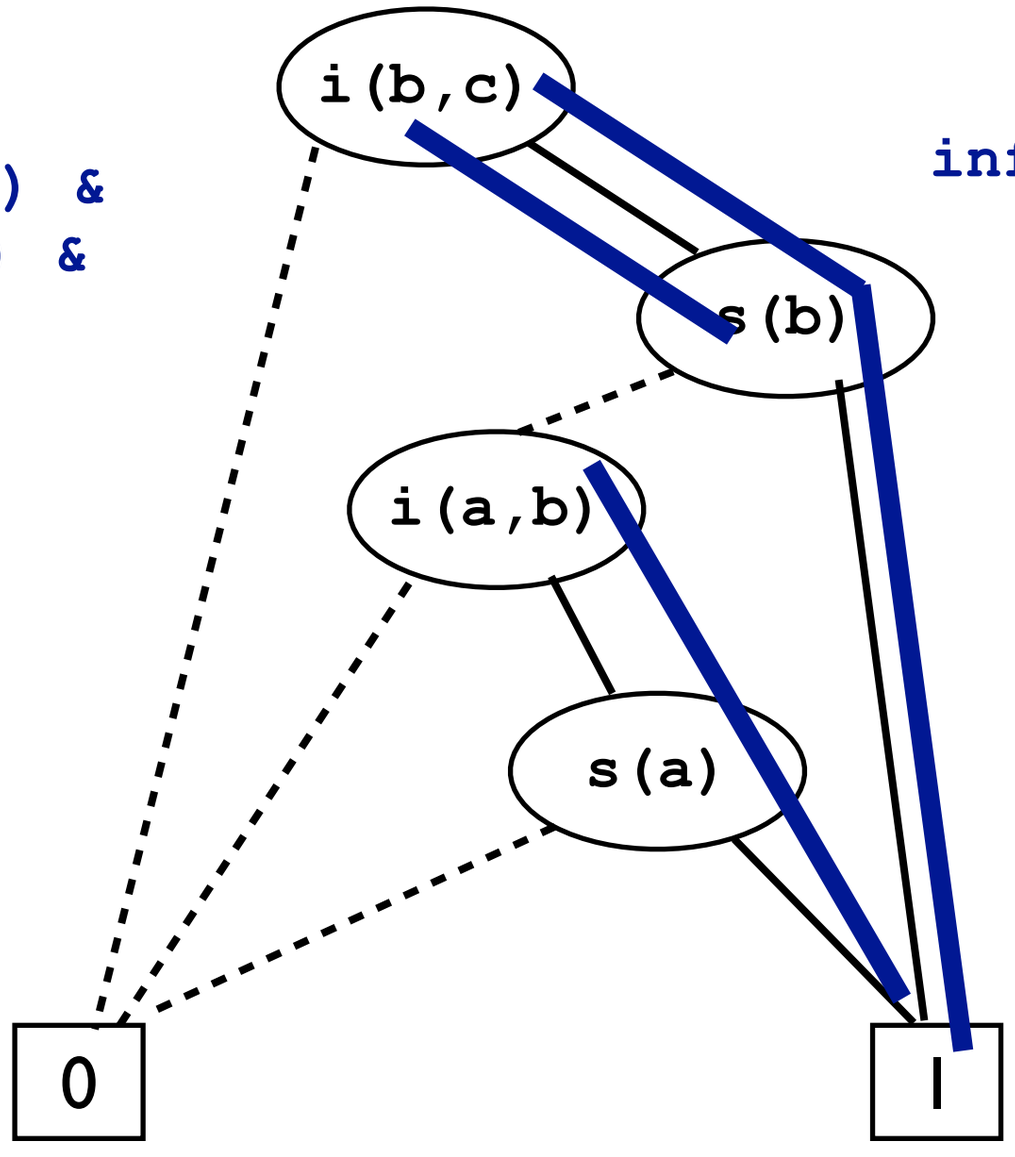
Binary Decision Diagrams
[Bryant 86]



influences(bob,carl) & stress(bob)

Binary Decision Diagrams
[Bryant 86]

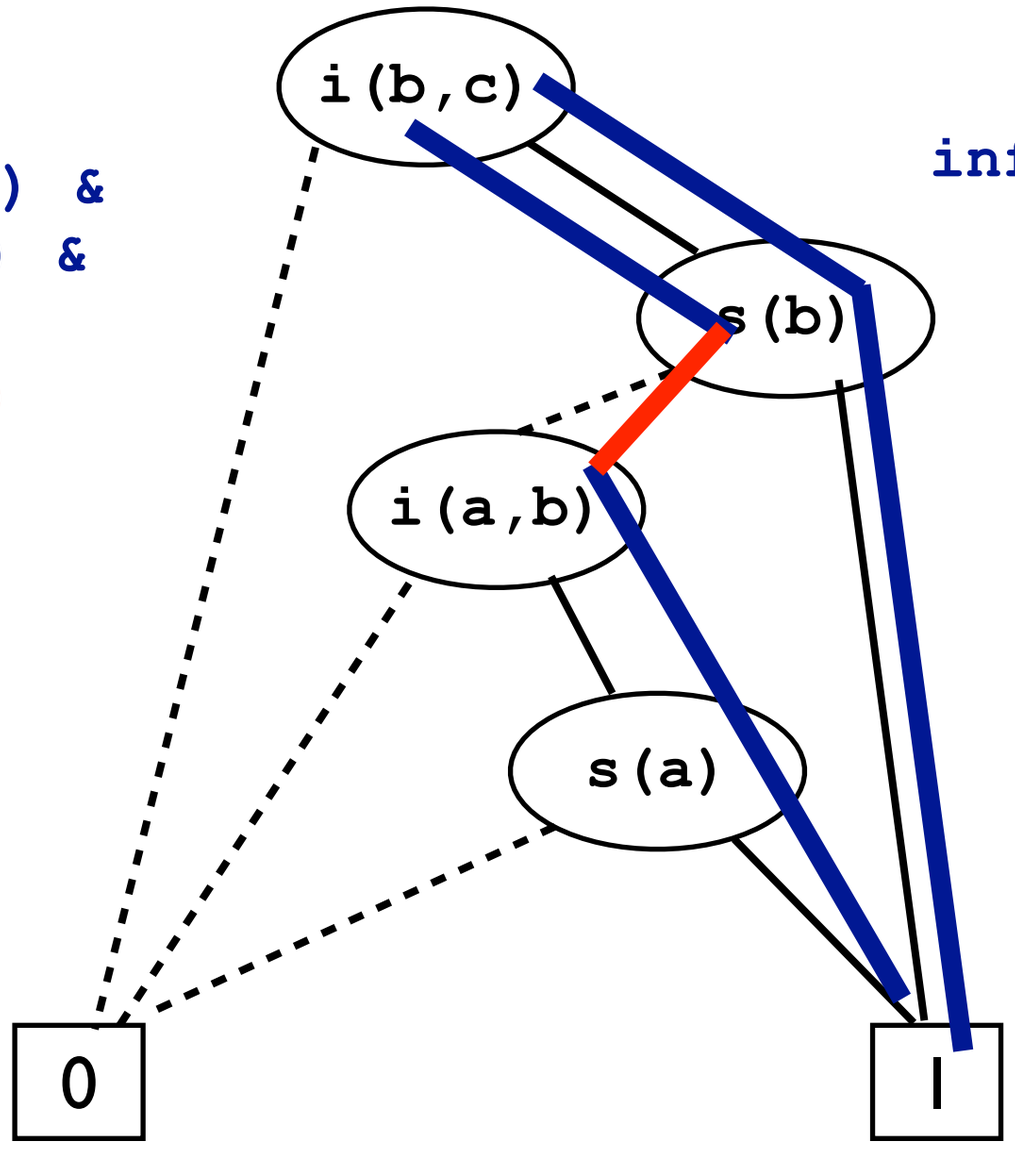
`influences(bob,carl) & influences(ann,bob) & stress(ann)`



`influences(bob,carl) & stress(bob)`

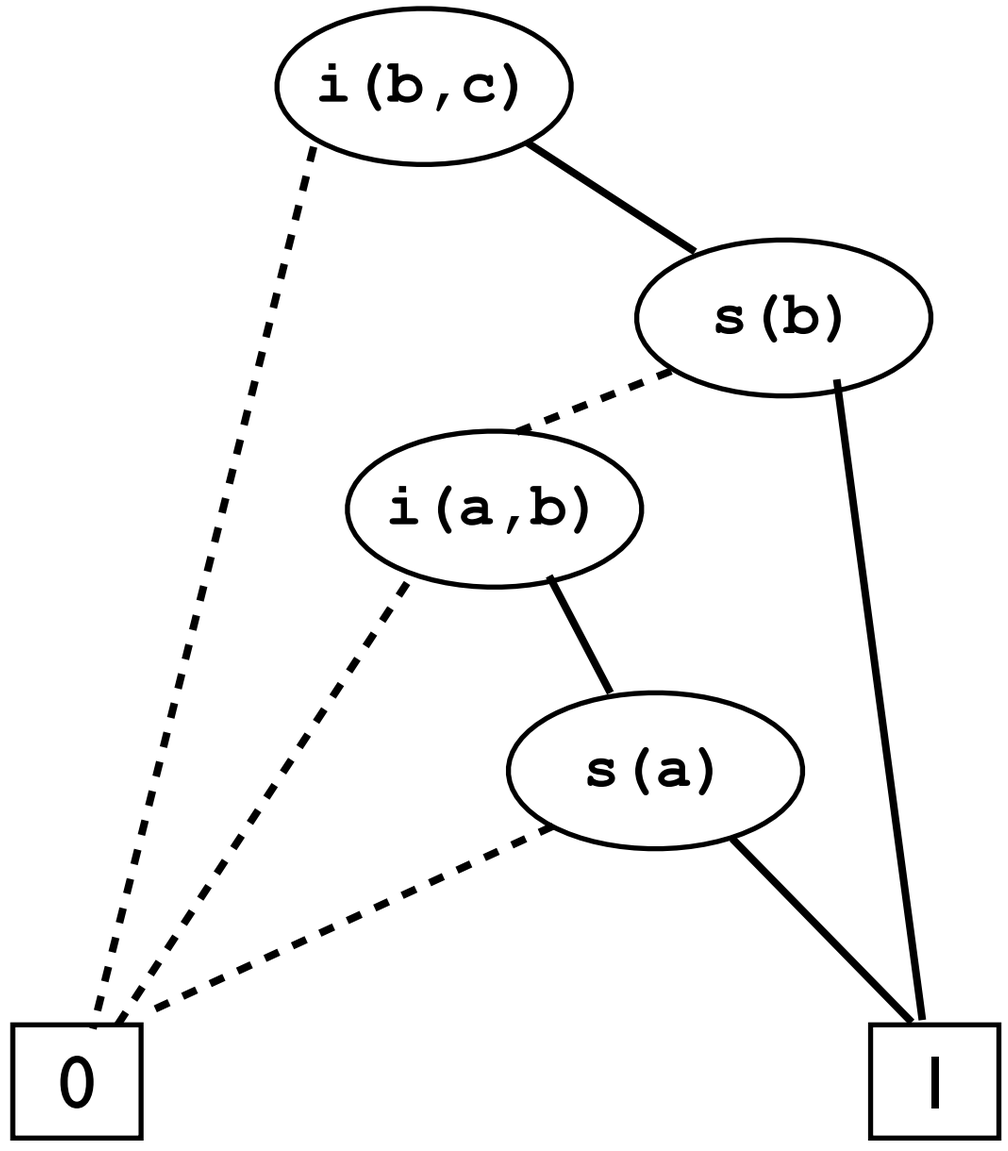
Binary Decision Diagrams
[Bryant 86]

influences(bob,carl) &
influences(ann,bob) &
stress(ann)
& not stress(bob)

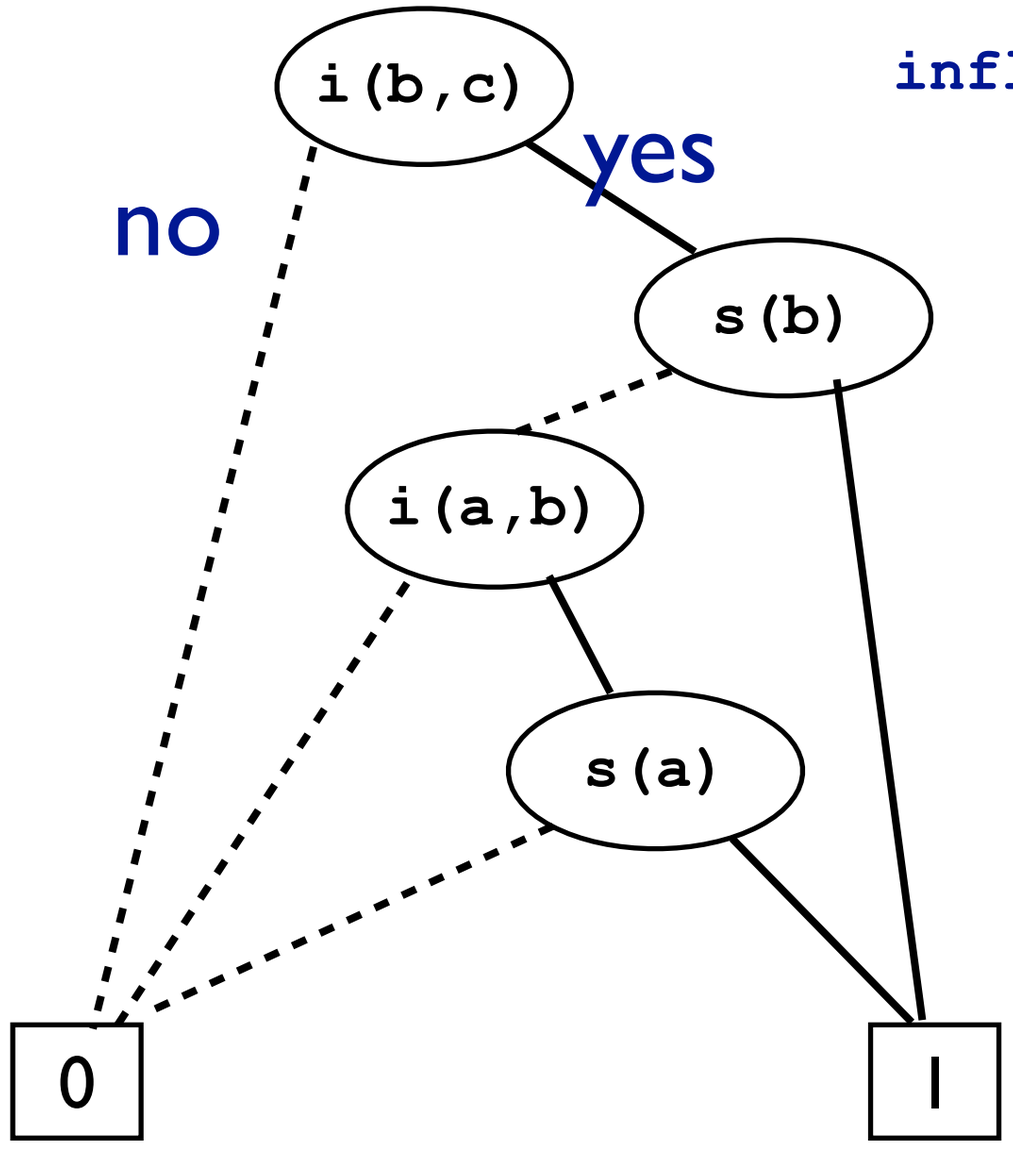


influences(bob,carl) &
stress(bob)

Binary Decision Diagrams

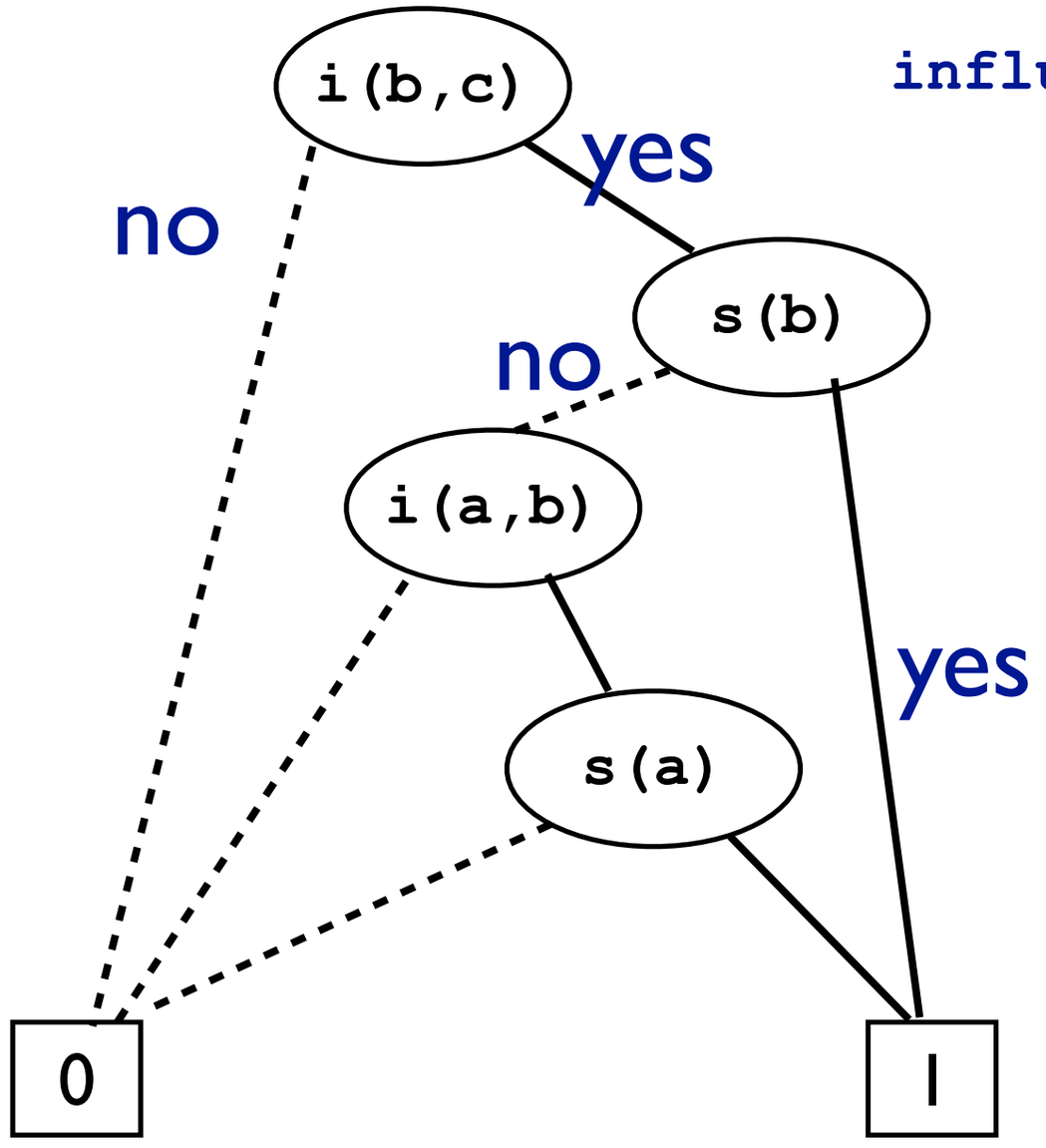


Binary Decision Diagrams



influences(bob,carl)?

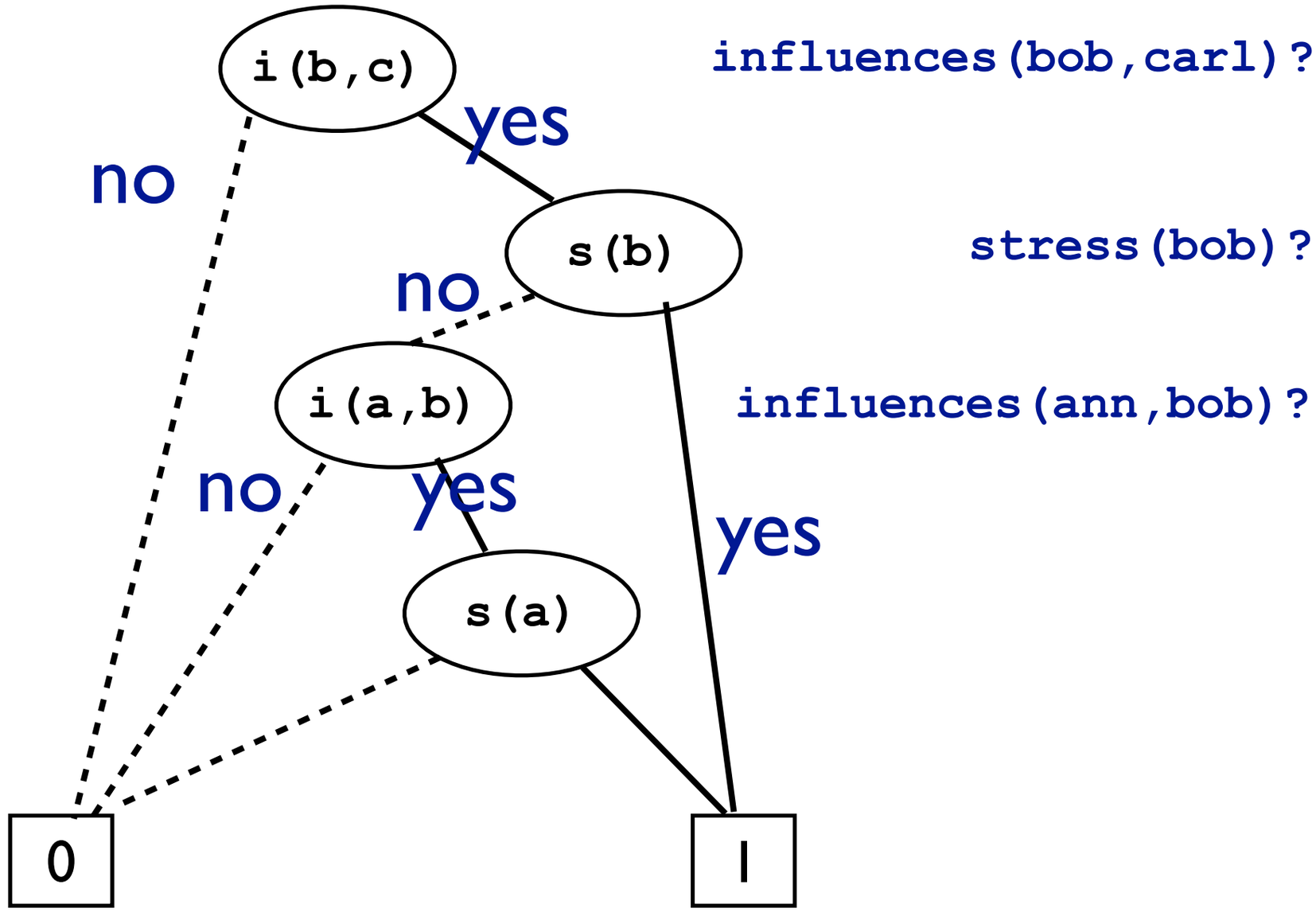
Binary Decision Diagrams



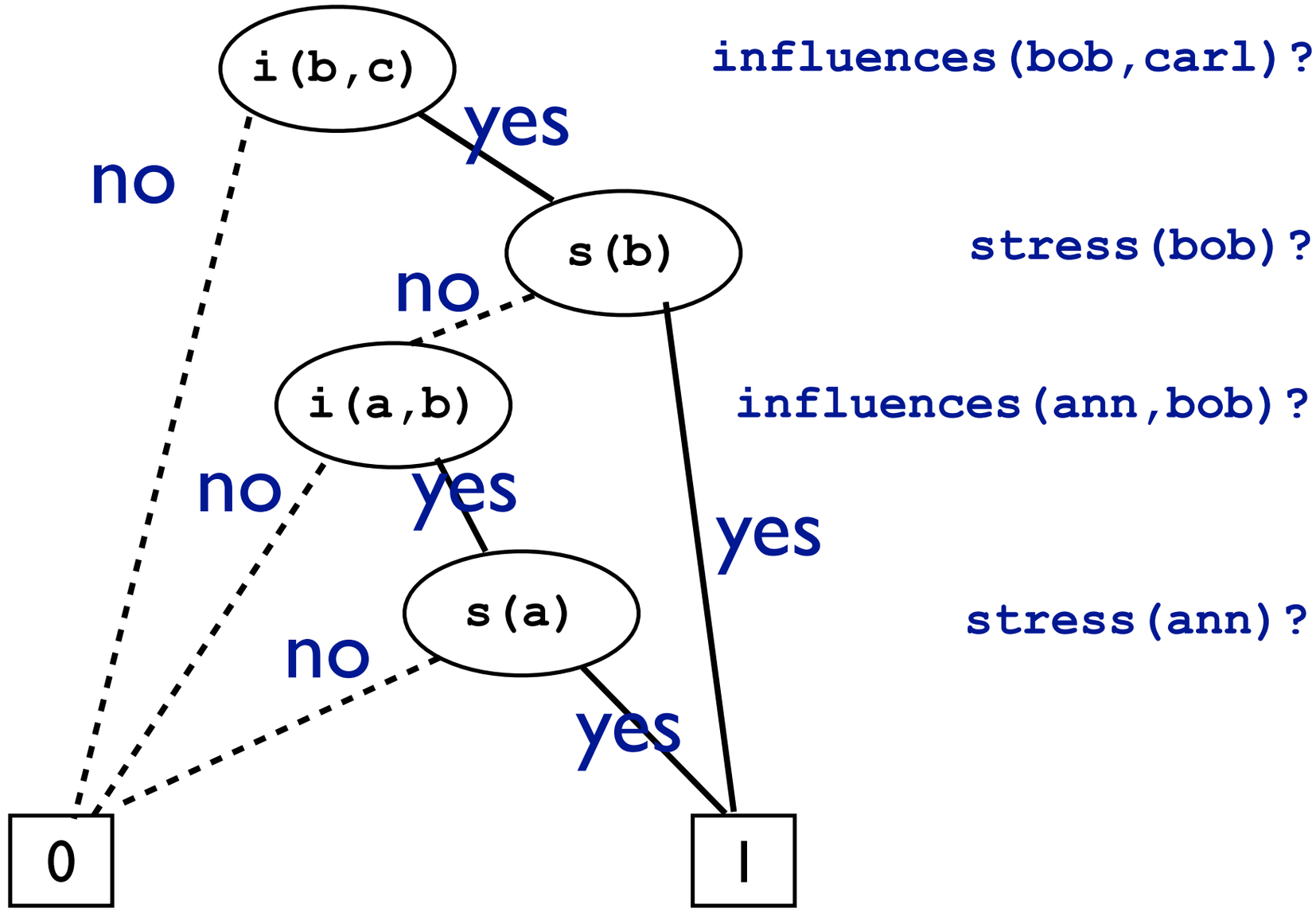
influences(bob,carl)?

stress(bob)?

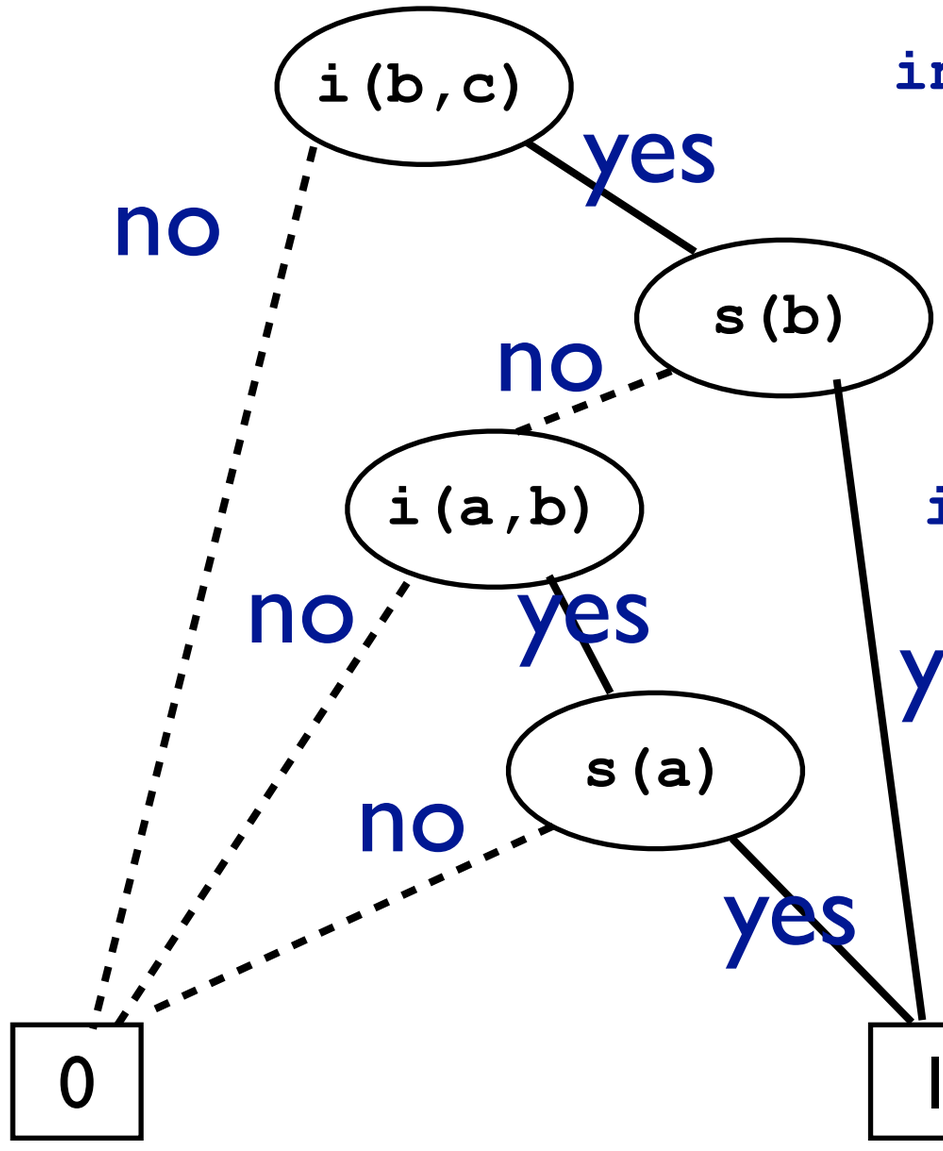
Binary Decision Diagrams



Binary Decision Diagrams



Binary Decision Diagrams



influences(bob,carl)?

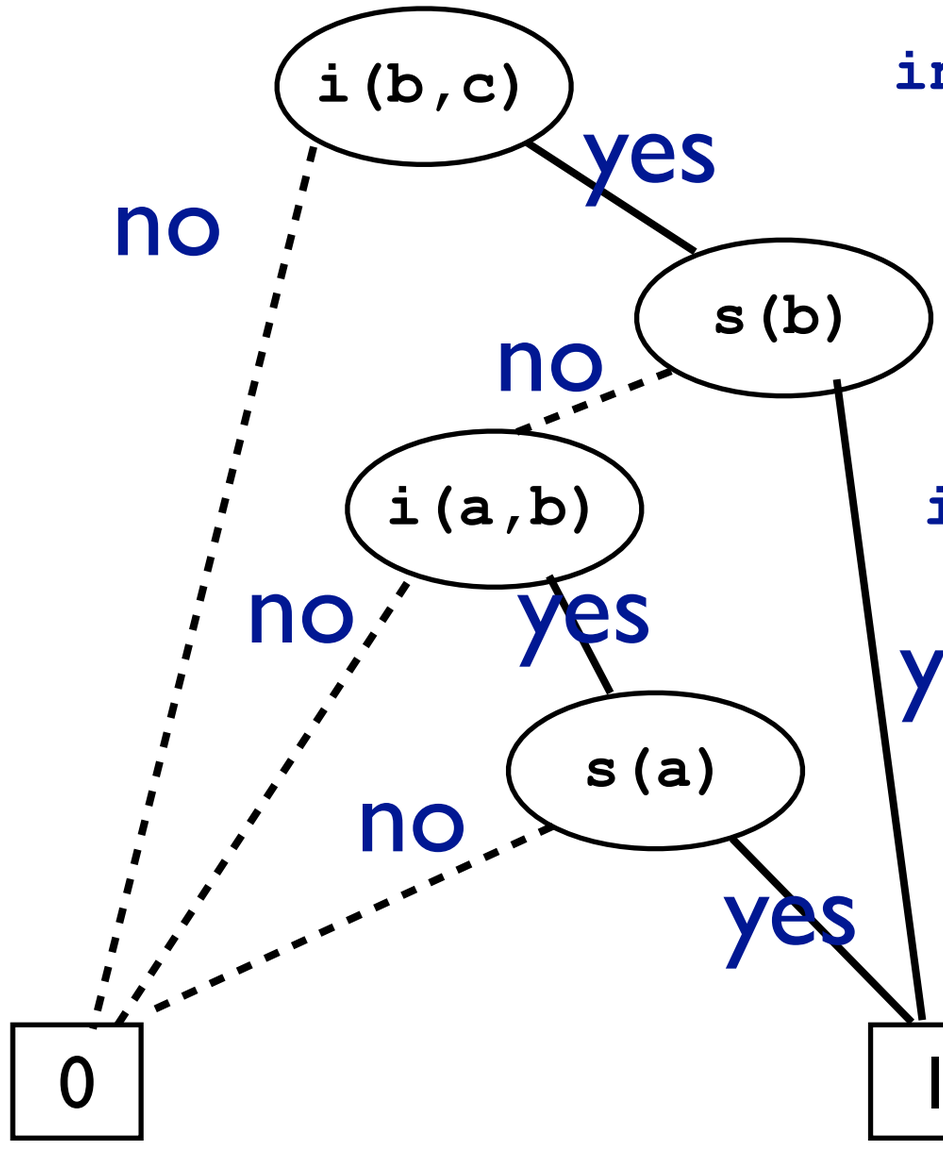
stress(bob)?

influences(ann,bob)?

stress(ann)?

smokes(c) = i(b,c)&s(b) v
i(b,c)&i(a,b)&s(a)

Binary Decision Diagrams



influences(bob,carl)?

stress(bob)?

influences(ann,bob)?

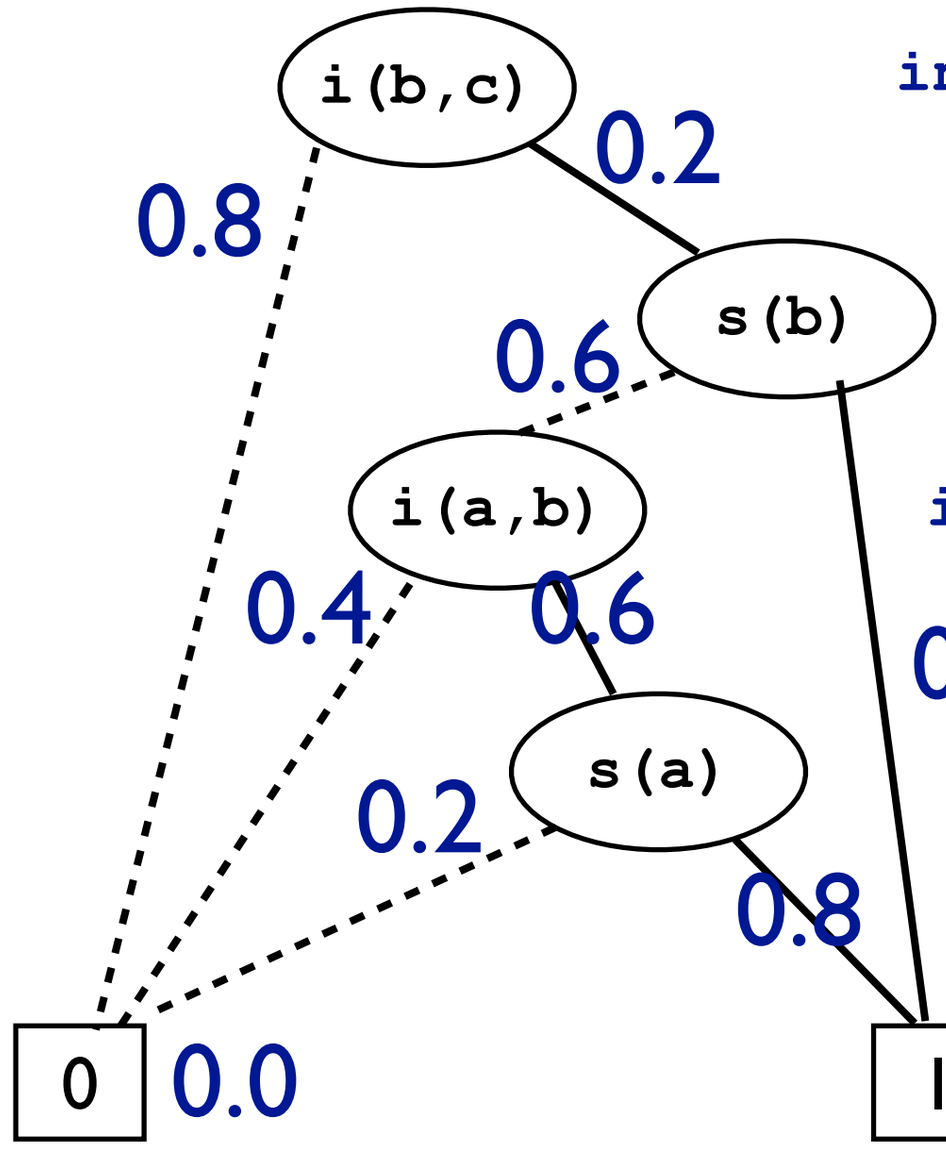
stress(ann)?

probability of

smokes(c)?

smokes(c) = i(b,c)&s(b) v
i(b,c)&i(a,b)&s(a)

Binary Decision Diagrams



`influences(bob,carl)?`

`stress(bob)?`

`influences(ann,bob)?`

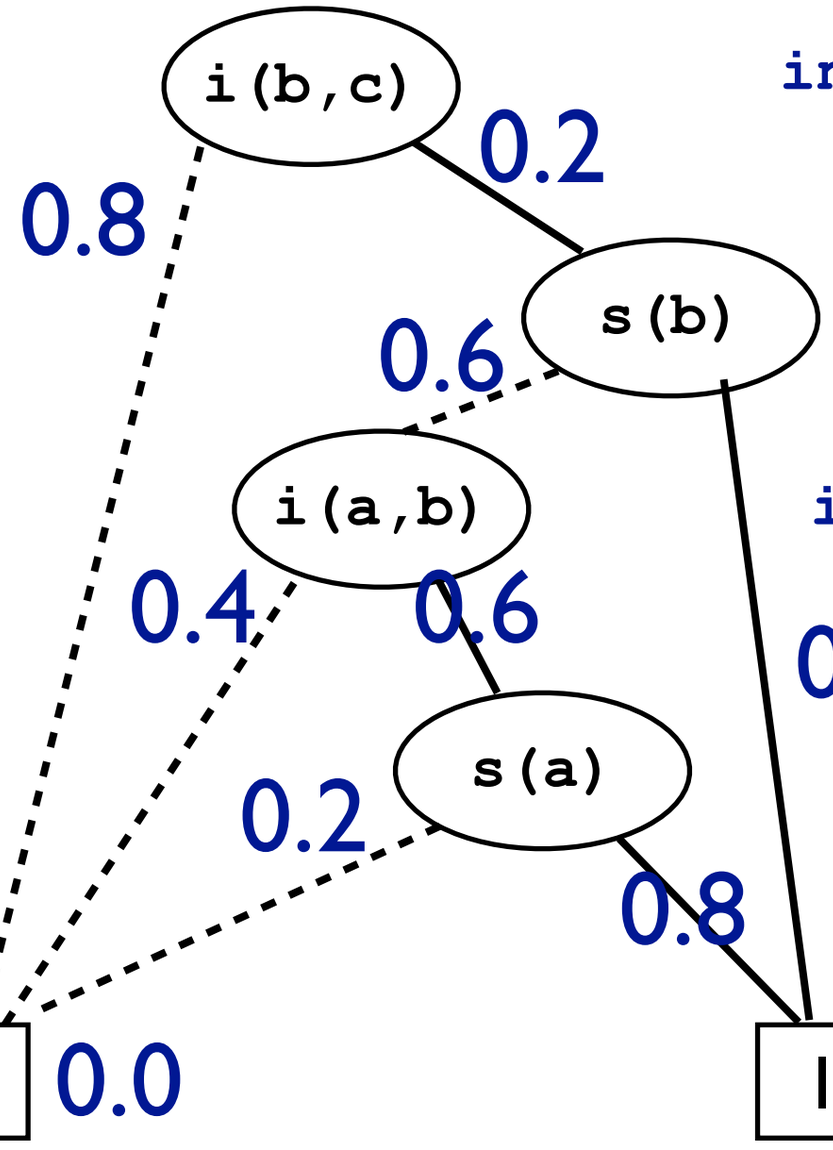
`stress(ann)?`

probability of

`smokes(c)?`

`smokes(c) = i(b,c)&s(b) v i(b,c)&i(a,b)&s(a)`

Binary Decision Diagrams



influences(bob,carl)?

stress(bob)?

influences(ann,bob)?

stress(ann)?

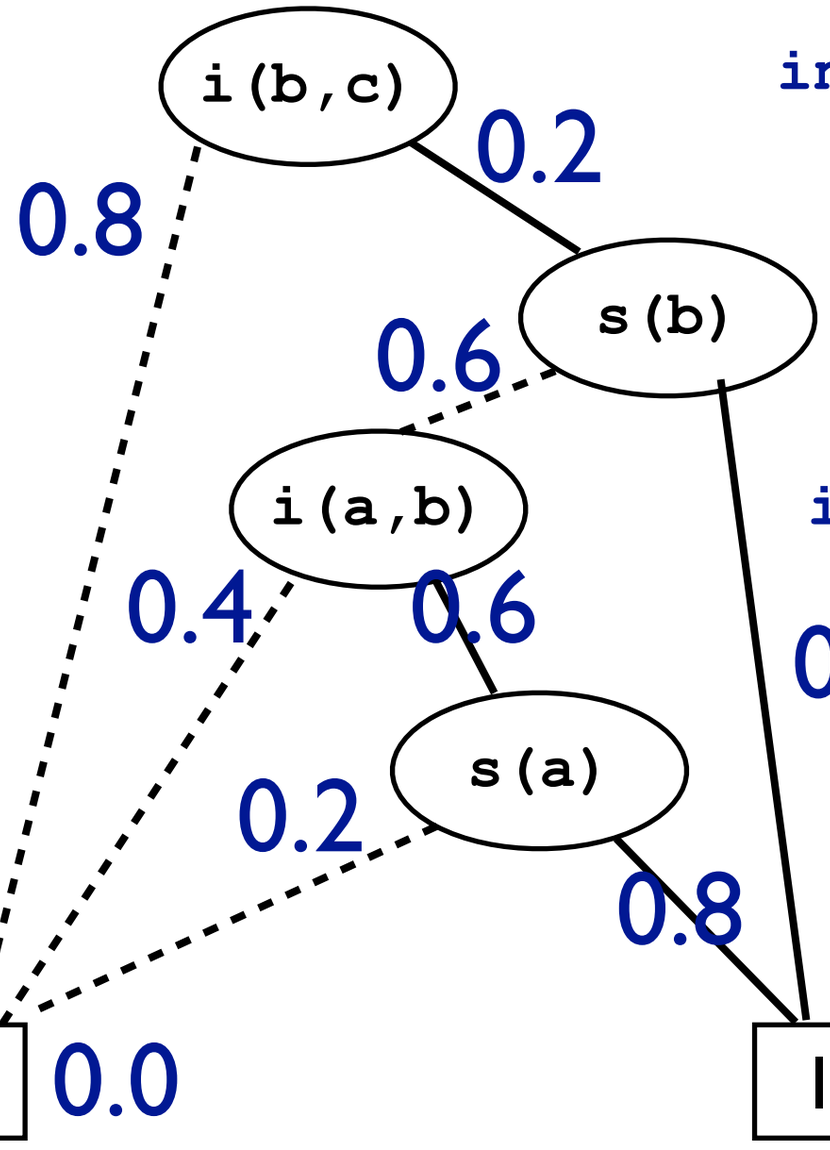
0.2×0.0+0.8×1.0=0.8

probability of

smokes(c)?

smokes(c) = i(b,c)&s(b) v
i(b,c)&i(a,b)&s(a)

Binary Decision Diagrams



`influences(bob,carl)?`

`stress(bob)?`

0.4×0.0+0.6×0.8=0.48

`influences(ann,bob)?`

0.2×0.0+0.8×1.0=0.8

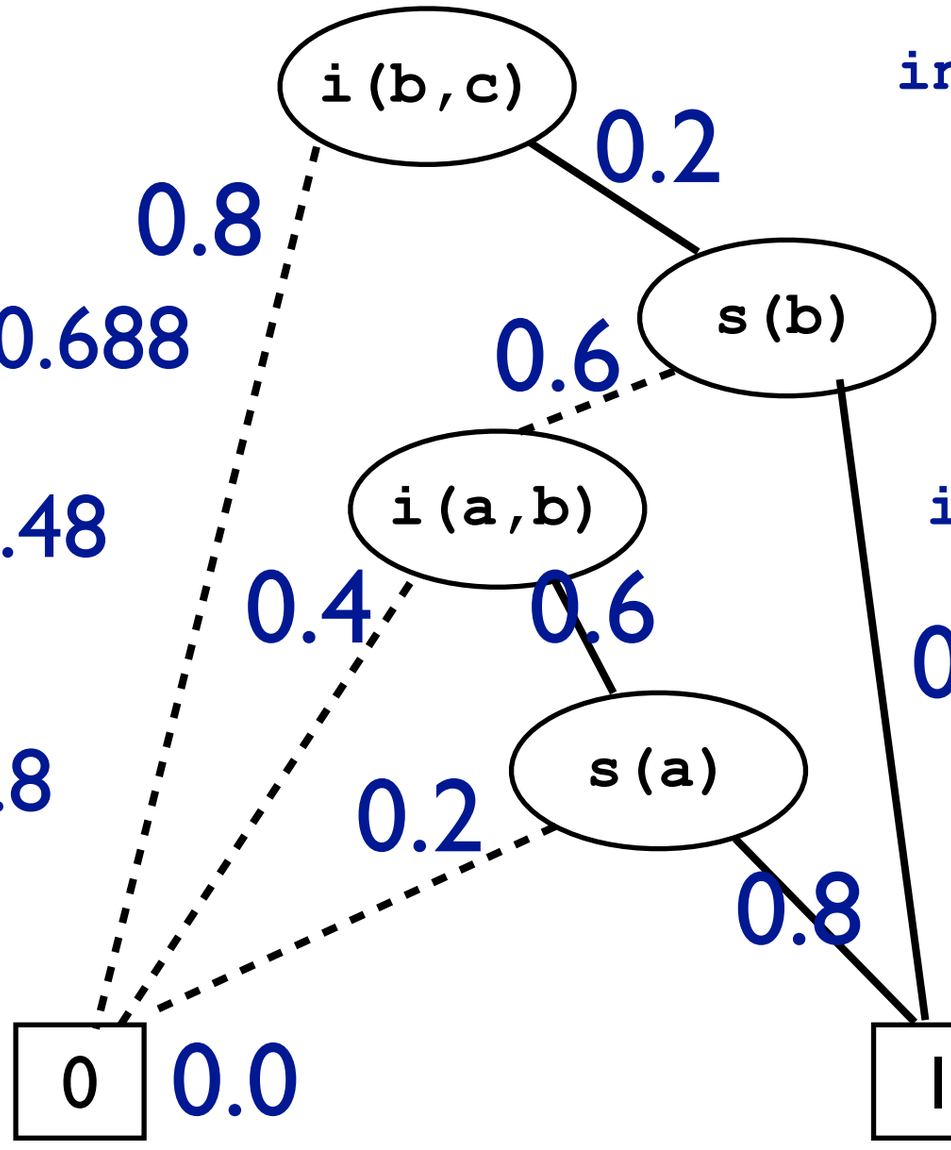
`stress(ann)?`

probability of

`smokes(c)?`

`smokes(c) = i(b,c)&s(b) v`
`i(b,c)&i(a,b)&s(a)`

Binary Decision Diagrams



influences(bob,carl)?

stress(bob)?

0.6×0.48+0.4×1.0=0.688

0.4×0.0+0.6×0.8=0.48

influences(ann,bob)?

0.2×0.0+0.8×1.0=0.8

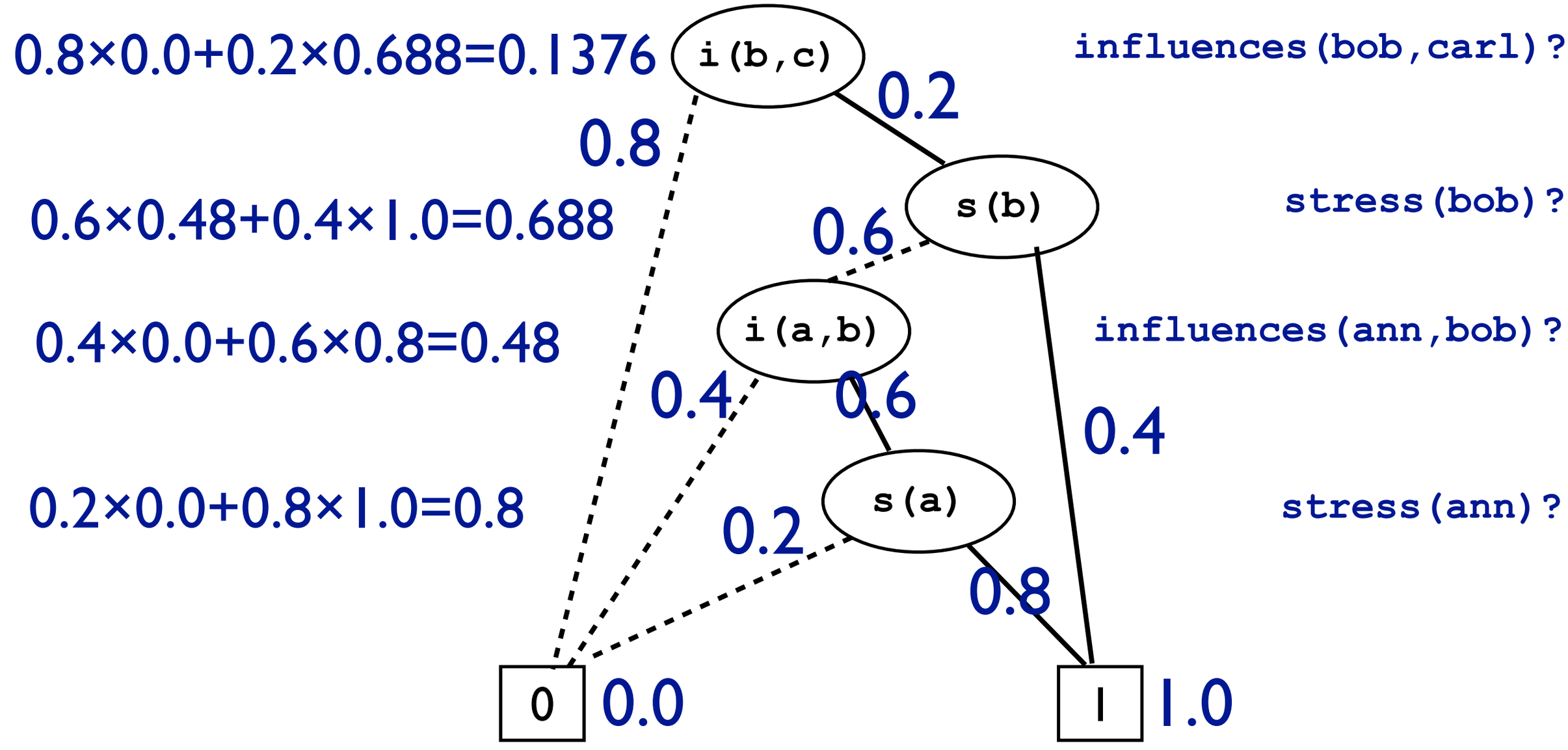
stress(ann)?

probability of

smokes(c)?

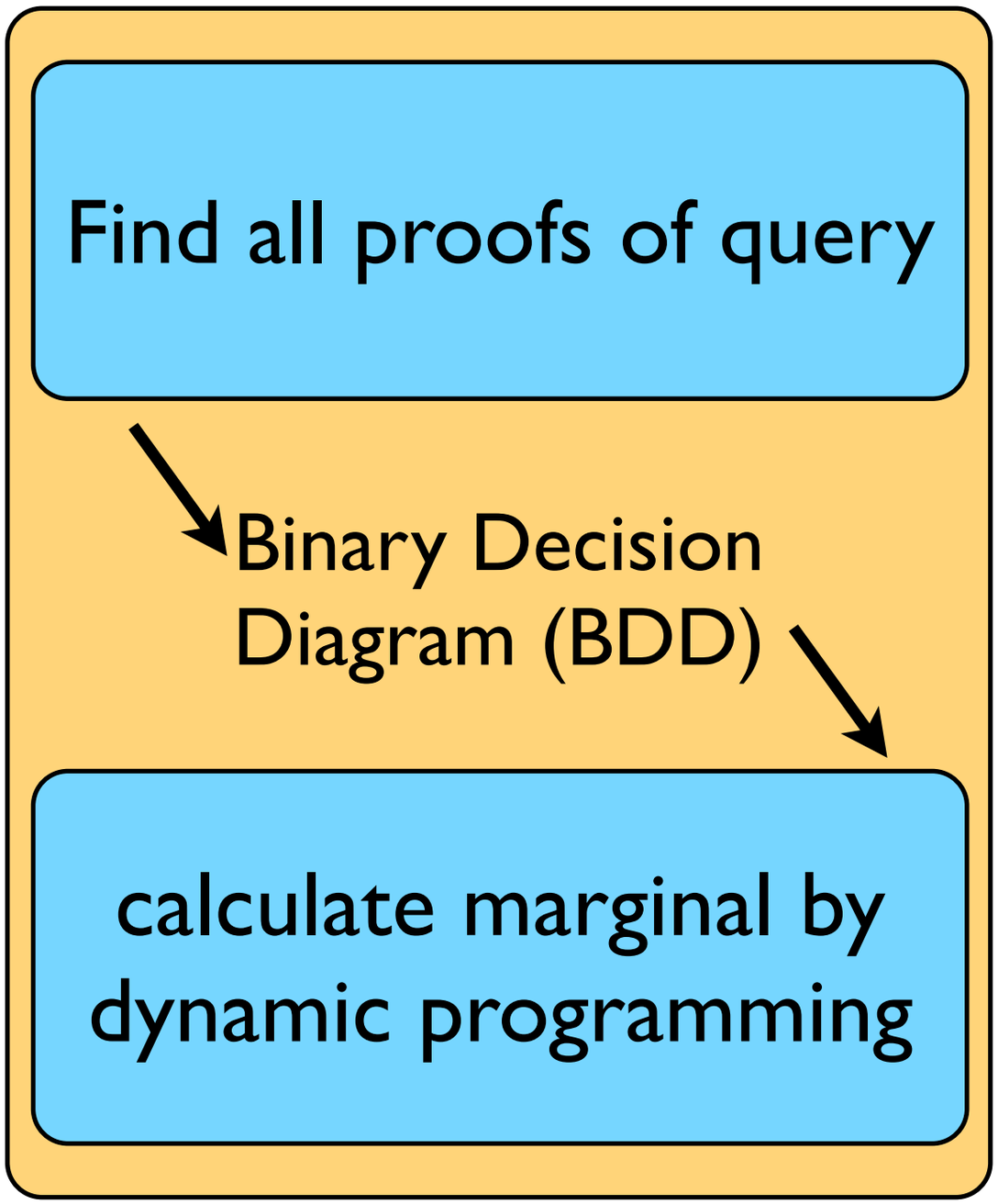
smokes(c) = i(b,c)&s(b) v
i(b,c)&i(a,b)&s(a)

Binary Decision Diagrams



Initial Approach

(ProbLog1 & others)

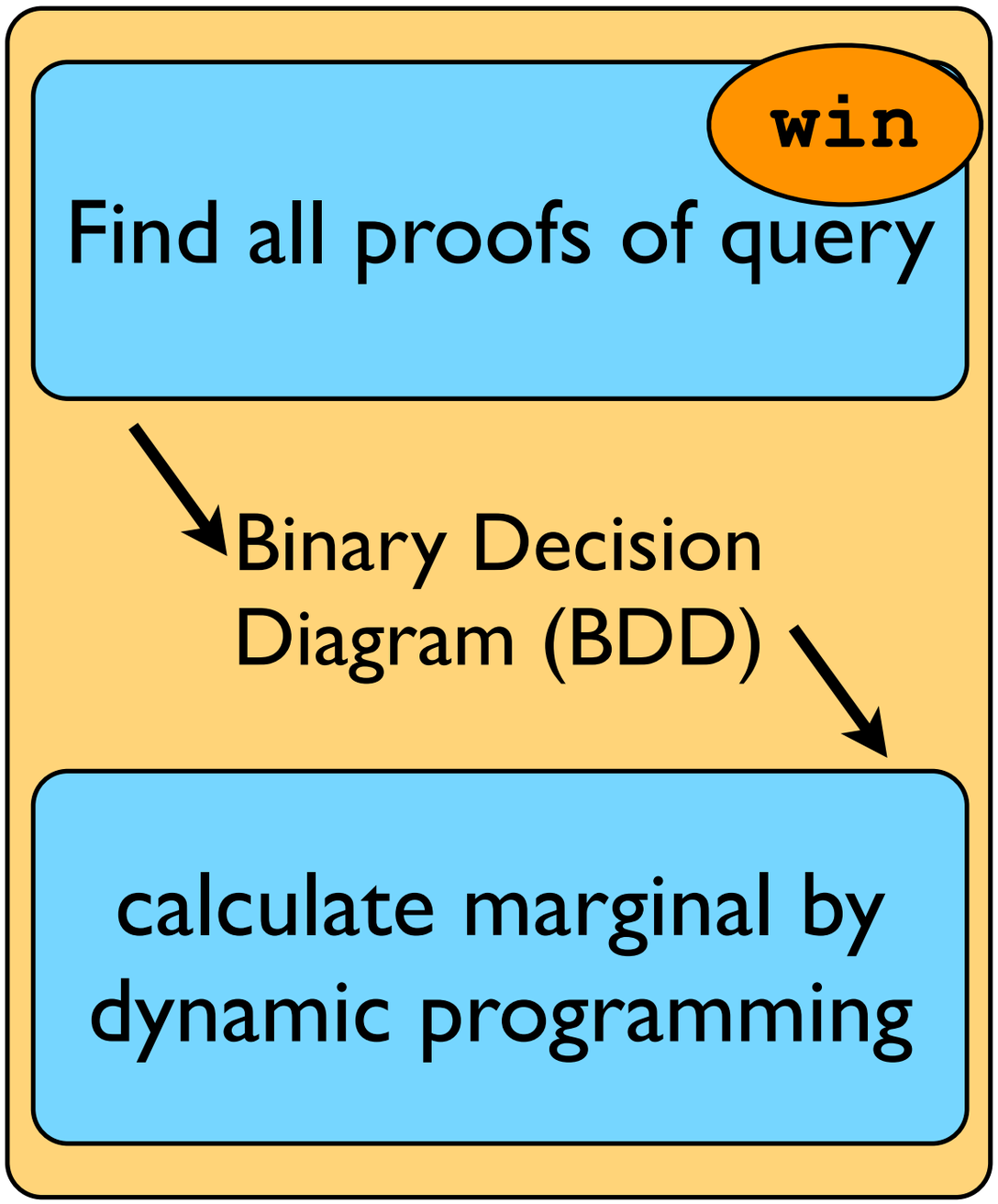


[De Raedt et al, IJCAI 07; Kimmig et al, TPLP 11]

Initial Approach

(ProbLog1 & others)

```
0.4::heads(1).
0.7::heads(2).
0.5::heads(3).
win :- heads(1).
win :- heads(2),heads(3).
```

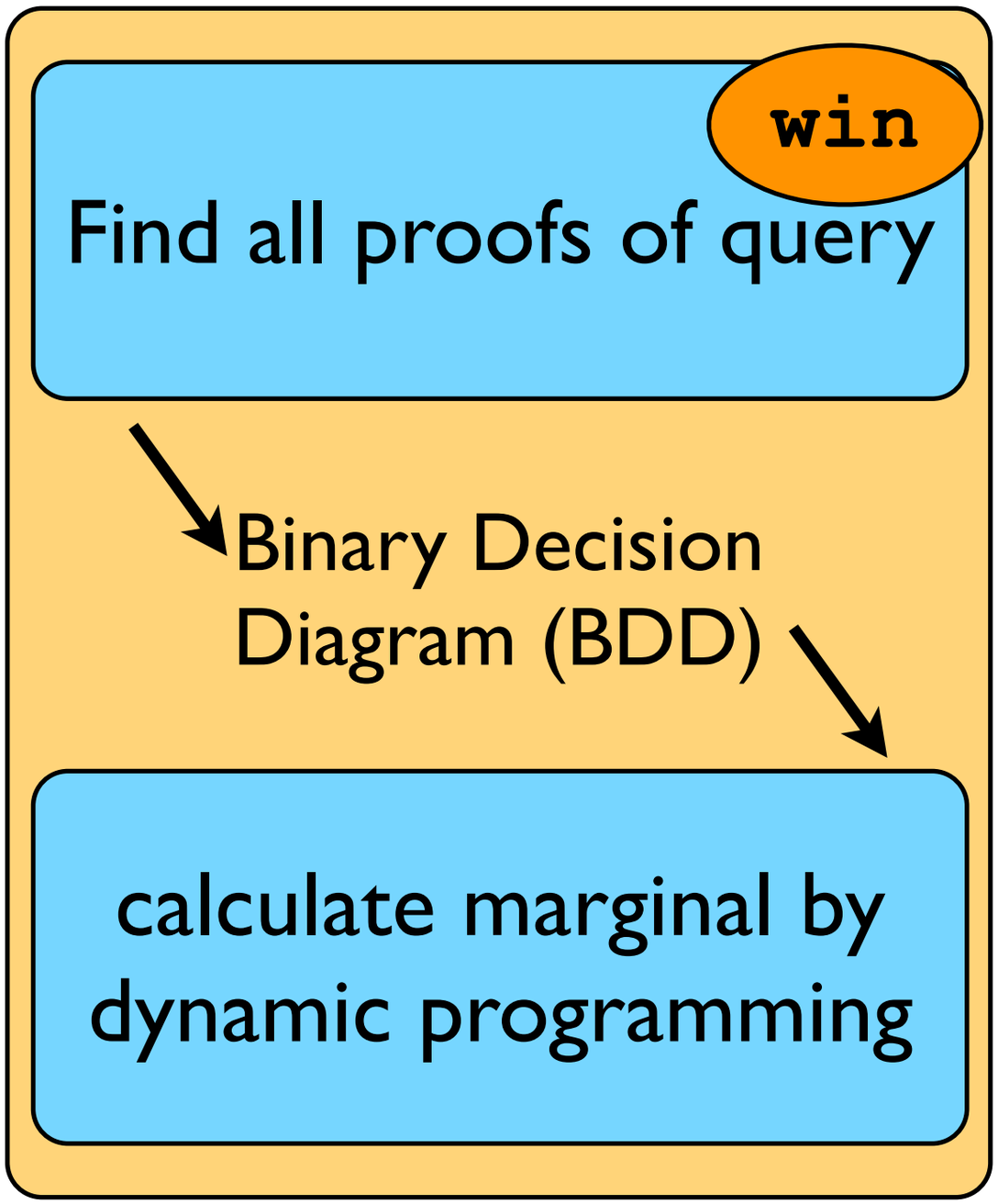


[De Raedt et al, IJCAI 07; Kimmig et al, TPLP 11]

Initial Approach

(ProbLog1 & others)

```
0.4::heads(1).
0.7::heads(2).
0.5::heads(3).
win :- heads(1).
win :- heads(2),heads(3).
```



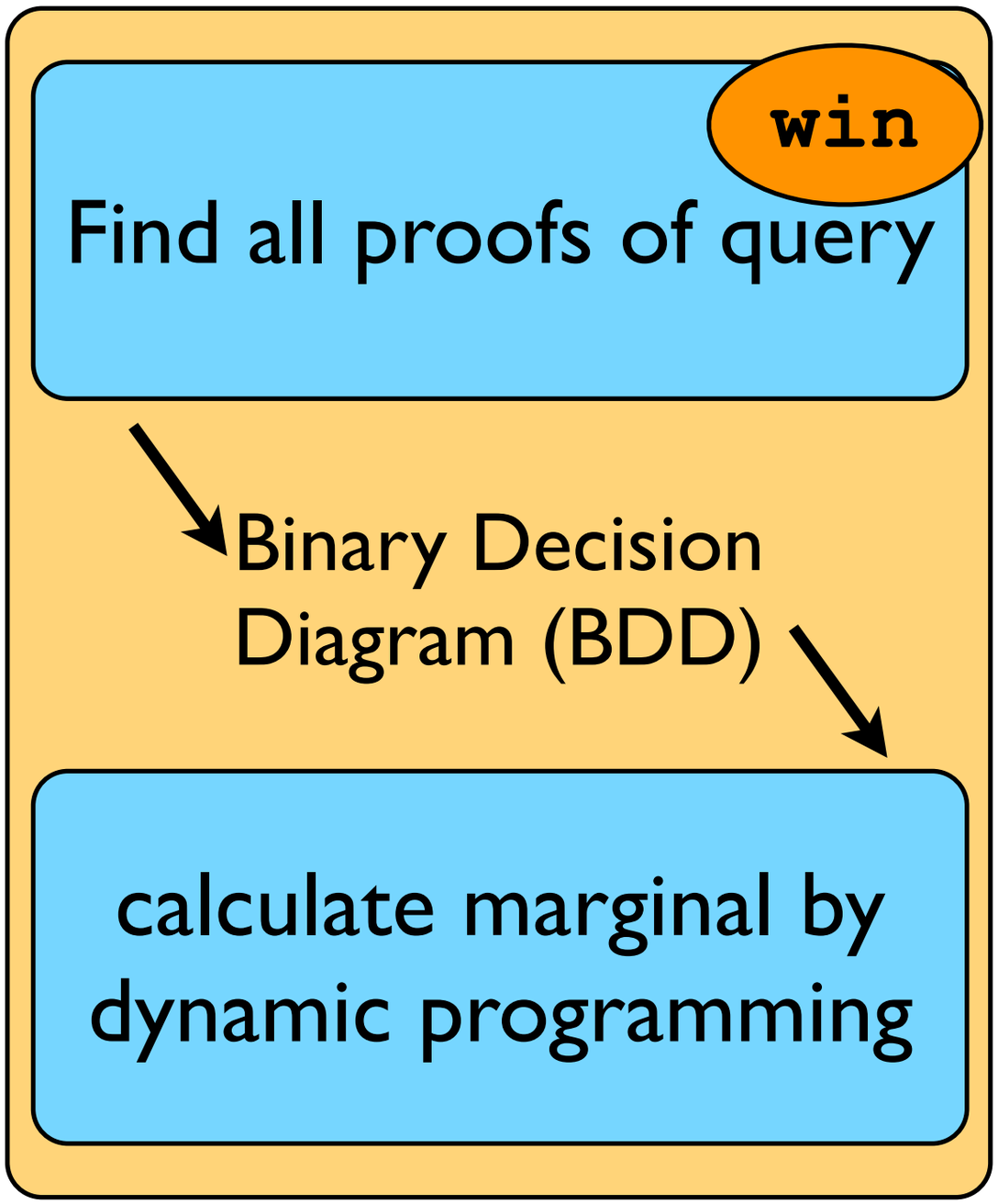

```
heads(1)
heads(2)  & heads(3)
```


[De Raedt et al, IJCAI 07; Kimmig et al, TPLP 11]

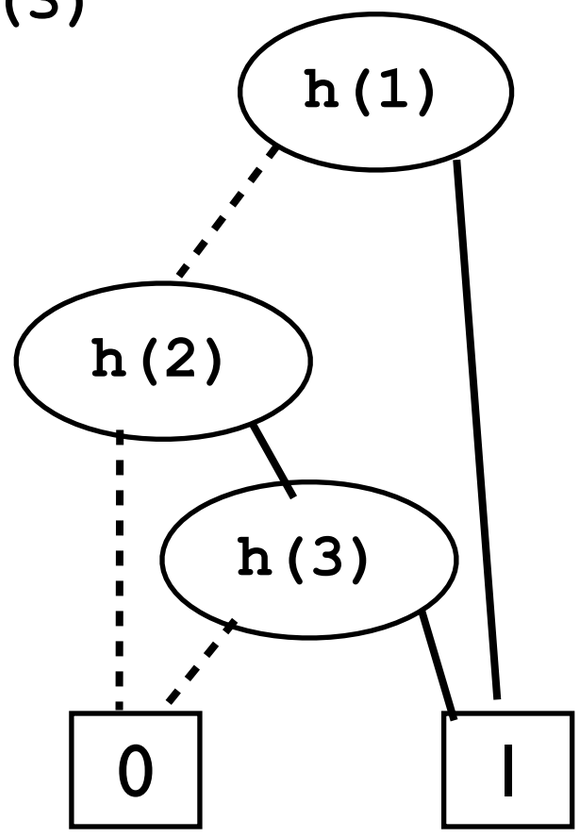
Initial Approach

(ProbLog1 & others)

```
0.4::heads(1).
0.7::heads(2).
0.5::heads(3).
win :- heads(1).
win :- heads(2),heads(3).
```




```
heads(1)
heads(2)  & heads(3)
```

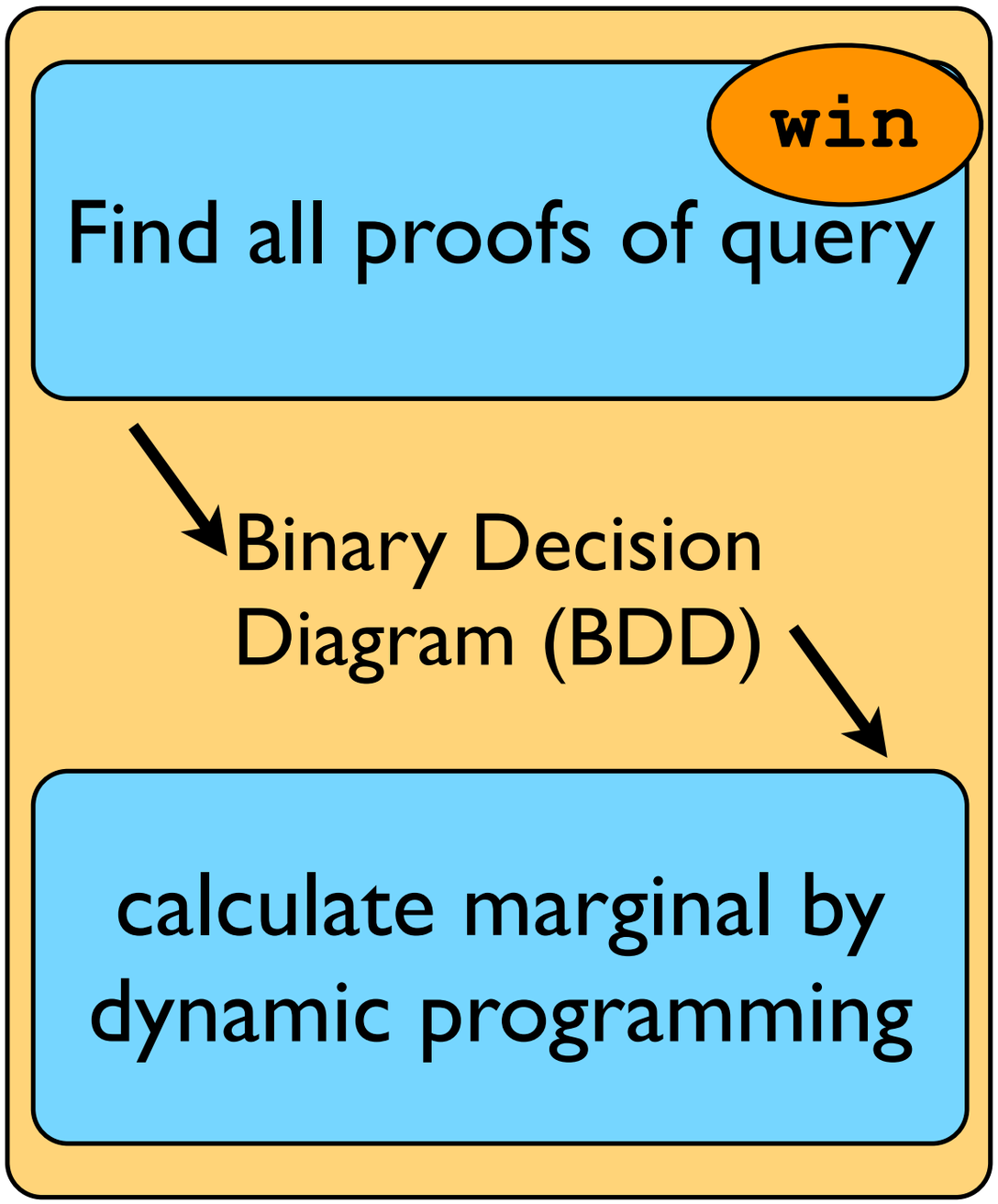


 [De Raedt et al, IJCAI 07; Kimmig et al, TPLP 11]

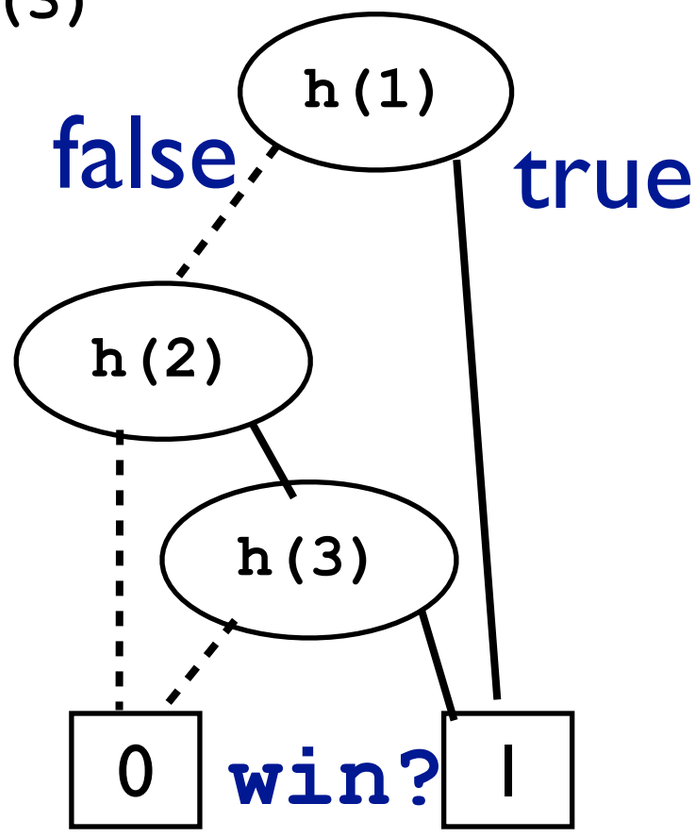
Initial Approach

(ProbLog1 & others)

```
0.4::heads(1).
0.7::heads(2).
0.5::heads(3).
win :- heads(1).
win :- heads(2),heads(3).
```




```
heads(1)
heads(2)  & heads(3)
```

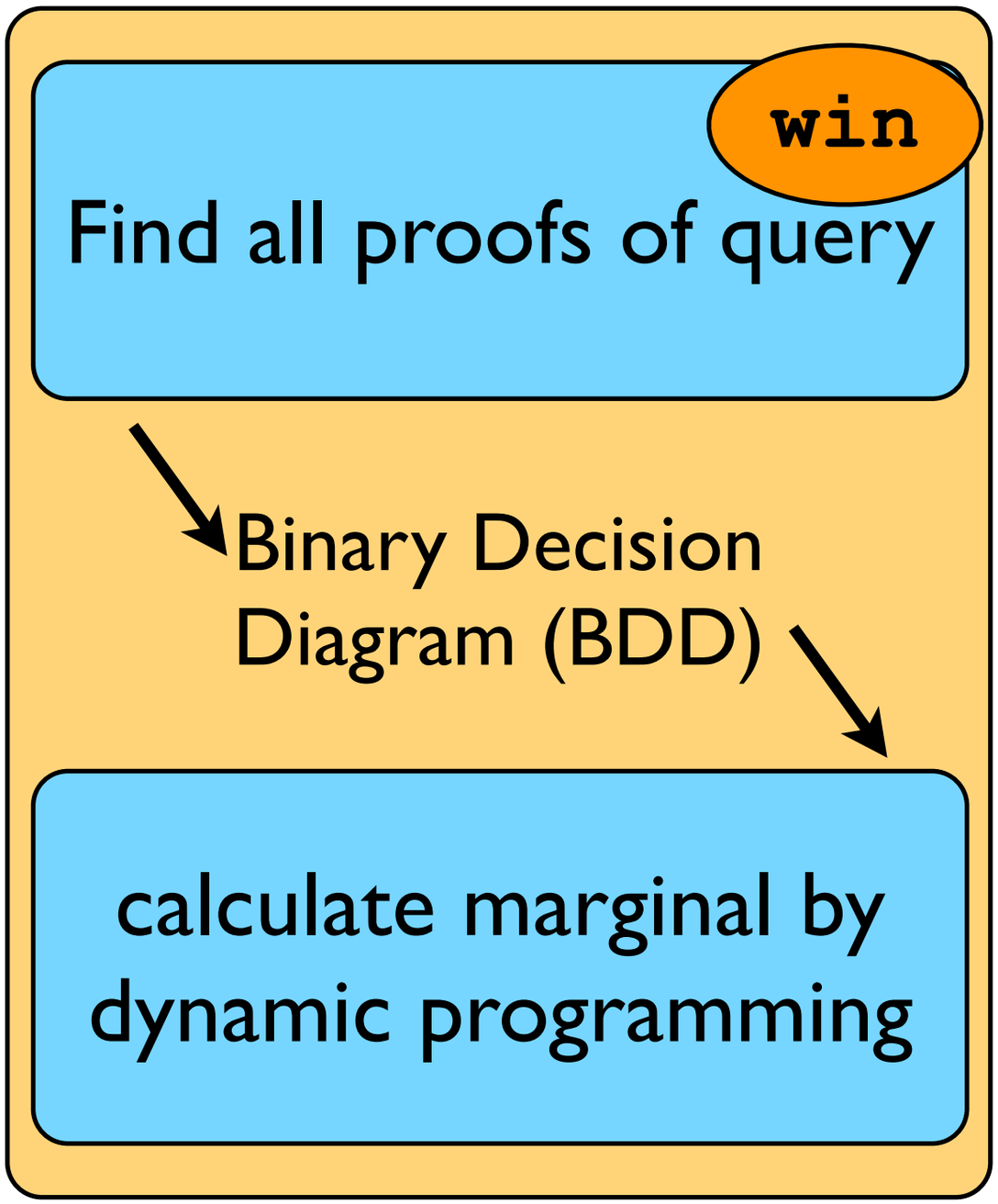


 [De Raedt et al, IJCAI 07; Kimmig et al, TPLP 11]

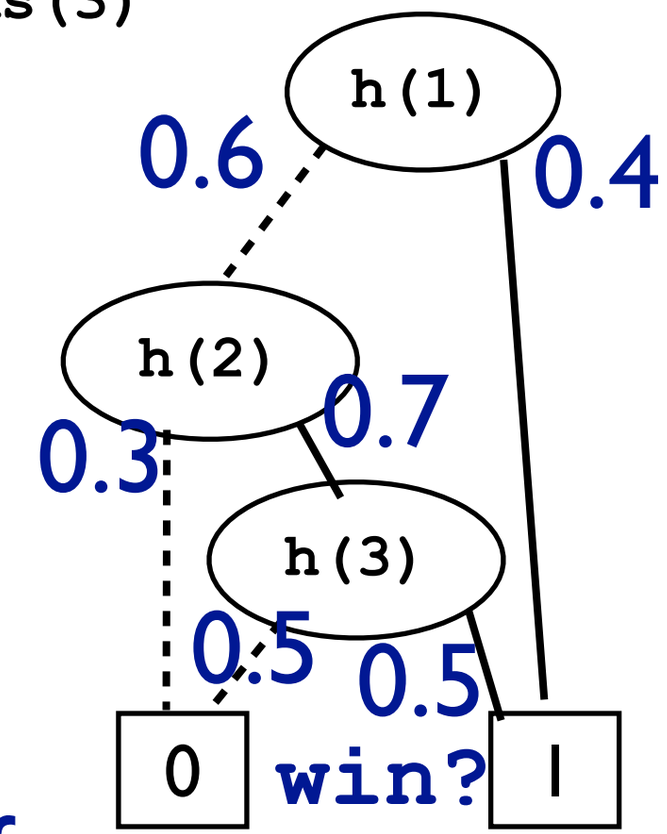
Initial Approach

(ProbLog1 & others)

```
0.4::heads(1).
0.7::heads(2).
0.5::heads(3).
win :- heads(1).
win :- heads(2),heads(3).
```




```
heads(1)
heads(2)  & heads(3)
```



P(**win**) = probability of reaching 1-leaf

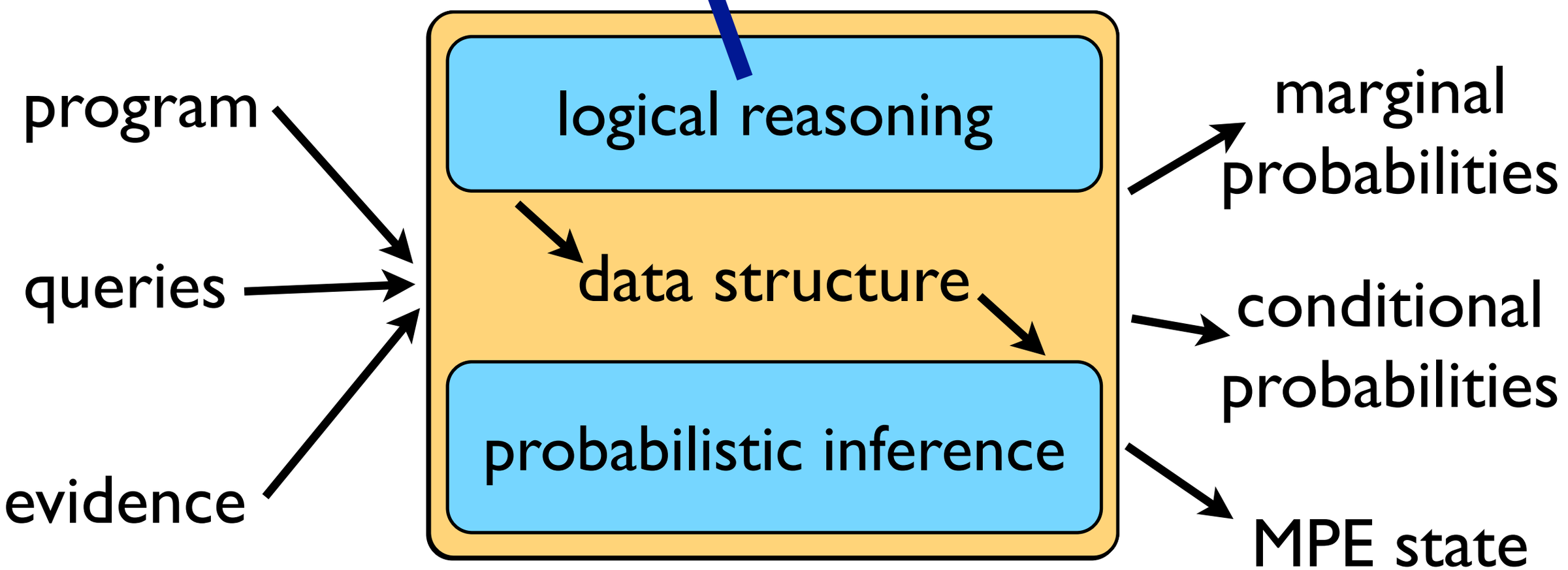
 [De Raedt et al, IJCAI 07; Kimmig et al, TPLP 11]

Answering Questions

1. using proofs
2. using models

Given:

Find:



Logical Reasoning: Models in Prolog

```
stress(ann).
influences(ann,bob).
influences(bob,carl).

smokes(X) :- stress(X).
smokes(X) :-
      influences(Y,X),
      smokes(Y).
```

```
?- smokes(carl).
```

- Forward reasoning to construct unique model:

Logical Reasoning: Models in Prolog

```
stress(ann).
influences(ann,bob).
influences(bob,carl).

smokes(X) :- stress(X).
smokes(X) :-
      influences(Y,X),
      smokes(Y).
```

```
?- smokes(carl).
```

- Forward reasoning to construct unique model:
  - Start with database facts

```
stress(ann).
influences(ann,bob).
influences(bob,carl).
```

Logical Reasoning: Models in Prolog

```
stress(ann).
influences(ann,bob).
influences(bob,carl).

smokes(X) :- stress(X).
smokes(X) :-
      influences(Y,X),
      smokes(Y).
```

```
?- smokes(carl).
```

- Forward reasoning to construct unique model:
  - Start with database facts
  - Use rules to add more facts

```
stress(ann).
influences(ann,bob).
influences(bob,carl).

smokes(ann).
```

Logical Reasoning: Models in Prolog

```
stress(ann).
influences(ann,bob).
influences(bob,carl).

smokes(X) :- stress(X).
smokes(X) :-
      influences(Y,X),
      smokes(Y).
```

```
?- smokes(carl).
```

- Forward reasoning to construct unique model:
  - Start with database facts

```
stress(ann).
influences(ann,bob).
influences(bob,carl).
```

  - Use rules to add more facts

```
smokes(ann).
smokes(bob).
```

Logical Reasoning: Models in Prolog

```
stress(ann).
influences(ann,bob).
influences(bob,carl).

smokes(X) :- stress(X).
smokes(X) :-
      influences(Y,X),
      smokes(Y).
```

```
?- smokes(carl).
```

- Forward reasoning to construct unique model:
  - Start with database facts

```
stress(ann).
influences(ann,bob).
influences(bob,carl).
```

  - Use rules to add more facts

```
smokes(ann).
smokes(bob).
smokes(carl).
```

Logical Reasoning: Models in Prolog

```
stress(ann).
influences(ann,bob).
influences(bob,carl).

smokes(X) :- stress(X).
smokes(X) :-
      influences(Y,X),
      smokes(Y).
```

```
?- smokes(carl).
```

- Forward reasoning to construct unique model:
  - Start with database facts
  - Use rules to add more facts
- Query true iff in model

```
stress(ann).
influences(ann,bob).
influences(bob,carl).

smokes(ann).
smokes(bob).
smokes(carl).
```

Logical Reasoning: Models in Prolog

```
stress(ann).
influences(ann,bob).
influences(bob,carl).

smokes(X) :- stress(X).
smokes(X) :-
      influences(Y,X),
      smokes(Y).
```

```
?- smokes(carl).
```

- Forward reasoning to construct unique model:
  - Start with database facts
  - Use rules to add more facts
- Query true iff in model

```
stress(ann).
influences(ann,bob).
influences(bob,carl).

smokes(ann).
smokes(bob).
smokes(carl).
```

- **ProbLog**: each possible world is a model, probability of query is sum over models where query is true

Logical Reasoning: Models in Prolog

```
stress(ann).
influences(ann,bob).
influences(bob,carl).

smokes(X) :- stress(X).
smokes(X) :-
      influences(Y,X),
      smokes(Y).
```

```
?- smokes(carl).
```

- Forward reasoning to construct unique model:
  - Start with database facts
  - Use rules to add more facts
- Query true iff in model

```
stress(ann).
influences(ann,bob).
influences(bob,carl).

smokes(ann).
smokes(bob).
smokes(carl).
```

- **ProbLog**: each possible world is a model, probability of query is sum over models where query is true

→ weighted model counting

Weighted Model Counting

$$WMC(\phi) = \sum_{I_V \models \phi} \prod_{l \in I} w(l)$$

Weighted Model Counting

propositional formula in conjunctive normal form (CNF)

$$WMC(\phi) = \sum_{I_V \models \phi} \prod_{l \in I} w(l)$$

Weighted Model Counting

propositional formula in conjunctive normal form (CNF)

$$WMC(\phi) = \sum_{I_V \models \phi} \prod_{l \in I} w(l)$$

interpretations (truth value assignments) of propositional variables

Weighted Model Counting

propositional formula in conjunctive normal form (CNF)

$$WMC(\phi) = \sum_{I_V \models \phi} \prod_{l \in I} w(l)$$

interpretations (truth value assignments) of propositional variables

weight of literal

Weighted Model Counting

propositional formula in conjunctive normal form (CNF)

given by ProbLog program & query

$$WMC(\phi) = \sum_{I_V \models \phi} \prod_{l \in I} w(l)$$

interpretations (truth value assignments) of propositional variables

weight of literal

Weighted Model Counting

propositional formula in conjunctive normal form (CNF)

given by ProbLog program & query

$$WMC(\phi) = \sum_{I_V \models \phi} \prod_{l \in I} w(l)$$

interpretations (truth value assignments) of propositional variables

possible worlds

weight of literal

Weighted Model Counting

propositional formula in conjunctive normal form (CNF)

given by ProbLog program & query

$$WMC(\phi) = \sum_{I_V \models \phi} \prod_{l \in I} w(l)$$

interpretations (truth value assignments) of propositional variables

possible worlds

weight of literal

for p::f,
w(f) = p
w(not f) = 1−p

Weighte

$$P(Q) = \sum_{F \cup R \models Q} \prod_{f \in F} p(f) \prod_{f \notin F} 1 - p(f)$$

propositional formula in conjunctive normal form (CNF)

given by ProbLog program & query

$$WMC(\phi) = \sum_{I_V \models \phi} \prod_{l \in I} w(l)$$

interpretations (truth value assignments) of propositional variables

possible worlds

weight of literal

for p::f,
w(f) = p
w(not f) = 1−p

ProbLog → CNF

```
?- smokes(carl).
```

```
0.8::stress(ann).
0.4::stress(bob).
0.6::influences(ann,bob).
0.2::influences(bob,carl).

smokes(X) :- stress(X).
smokes(X) :-
      influences(Y,X),
      smokes(Y).
```

ProbLog → CNF

```
0.8::stress(ann).
0.4::stress(bob).
0.6::influences(ann,bob).
0.2::influences(bob,carl).

smokes(X) :- stress(X).
smokes(X) :-
      influences(Y,X),
      smokes(Y).
```

```
?- smokes(carl).
```

- Find relevant ground rules by backward reasoning

ProbLog →CNF

```
0.8::stress(ann).
0.4::stress(bob).
0.6::influences(ann,bob).
0.2::influences(bob,carl).

smokes(X) :- stress(X).
smokes(X) :-
     influences(Y,X),
     smokes(Y).
```

```
?- smokes(carl).
```

- Find relevant ground rules by backward reasoning

```
smokes(carl) :- influences(bob,carl),smokes(bob).
smokes(bob) :- stress(bob).
smokes(bob) :- influences(ann,bob),smokes(ann).
smokes(ann) :- stress(ann).
```

ProbLog →CNF

```
0.8::stress(ann).
0.4::stress(bob).
0.6::influences(ann,bob).
0.2::influences(bob,carl).

smokes(X) :- stress(X).
smokes(X) :-
     influences(Y,X),
     smokes(Y).
```

```
?- smokes(carl).
```

- Find relevant ground rules by backward reasoning

```
smokes(carl) :- influences(bob,carl),smokes(bob).
smokes(bob) :- stress(bob).
smokes(bob) :- influences(ann,bob),smokes(ann).
smokes(ann) :- stress(ann).
```

- Convert to propositional logic formula

ProbLog → CNF

```
0.8::stress(ann).
0.4::stress(bob).
0.6::influences(ann,bob).
0.2::influences(bob,carl).

smokes(X) :- stress(X).
smokes(X) :-
     influences(Y,X),
     smokes(Y).
```

```
?- smokes(carl).
```

- Find relevant ground rules by backward reasoning

```
smokes(carl) :- influences(bob,carl),smokes(bob).
smokes(bob) :- stress(bob).
smokes(bob) :- influences(ann,bob),smokes(ann).
smokes(ann) :- stress(ann).
```

- Convert to propositional logic formula

$$sm(c) \leftrightarrow (i(b,c) \wedge sm(b))$$
$$\wedge\ sm(b) \leftrightarrow (st(b) \vee (i(a,b) \wedge sm(a)))$$
$$\wedge\ sm(a) \leftrightarrow st(a)$$

ProbLog → CNF

```
0.8::stress(ann).
0.4::stress(bob).
0.6::influences(ann,bob).
0.2::influences(bob,carl).

smokes(X) :- stress(X).
smokes(X) :-
     influences(Y,X),
     smokes(Y).
```

```
?- smokes(carl).
```

- Find relevant ground rules by backward reasoning

```
smokes(carl) :- influences(bob,carl),smokes(bob).
smokes(bob) :- stress(bob).
smokes(bob) :- influences(ann,bob),smokes(ann).
smokes(ann) :- stress(ann).
```

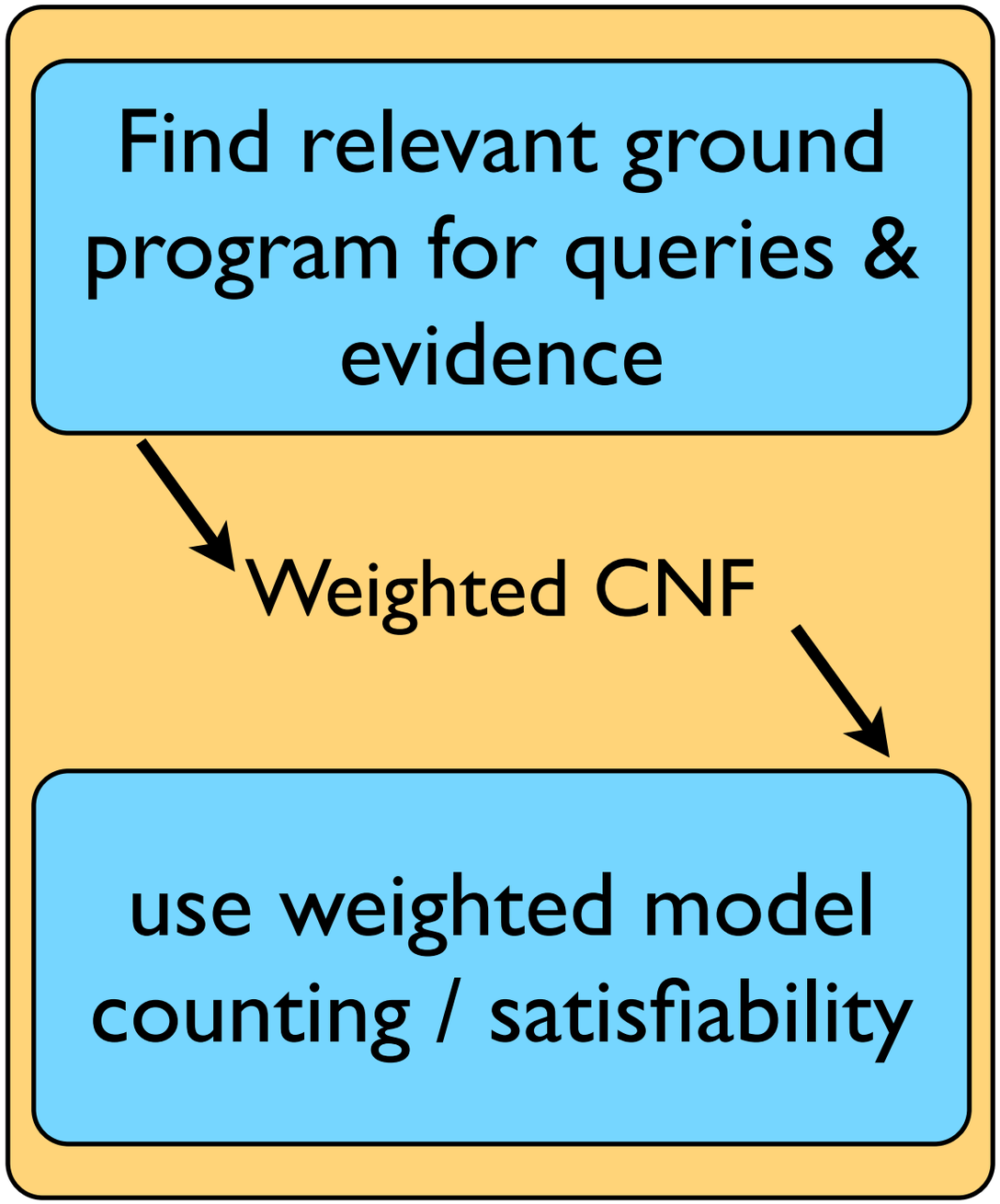
- Convert to propositional logic formula

$$sm(c) \leftrightarrow (i(b,c) \wedge sm(b))$$
$$\wedge\ sm(b) \leftrightarrow (st(b) \vee (i(a,b) \wedge sm(a)))$$
$$\wedge\ sm(a) \leftrightarrow st(a)$$

- Rewrite in CNF (as usual)

Current Approach

(ProbLog2)

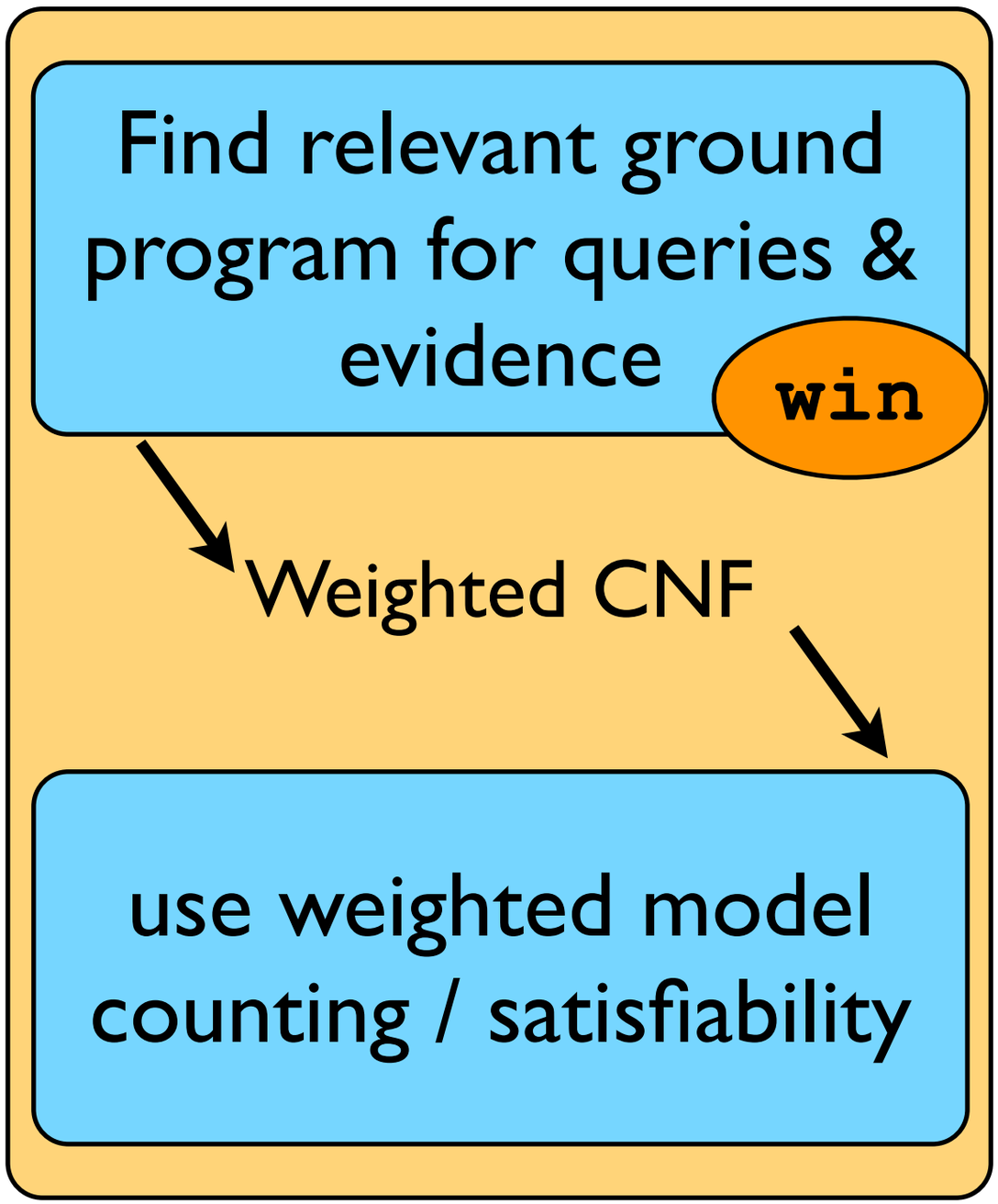


[Fierens et al, TPLP 14]

Current Approach

(ProbLog2)

```
0.4::heads(1).
0.7::heads(2).
0.5::heads(3).
win :- heads(1).
win :- heads(2),
       heads(3).
```

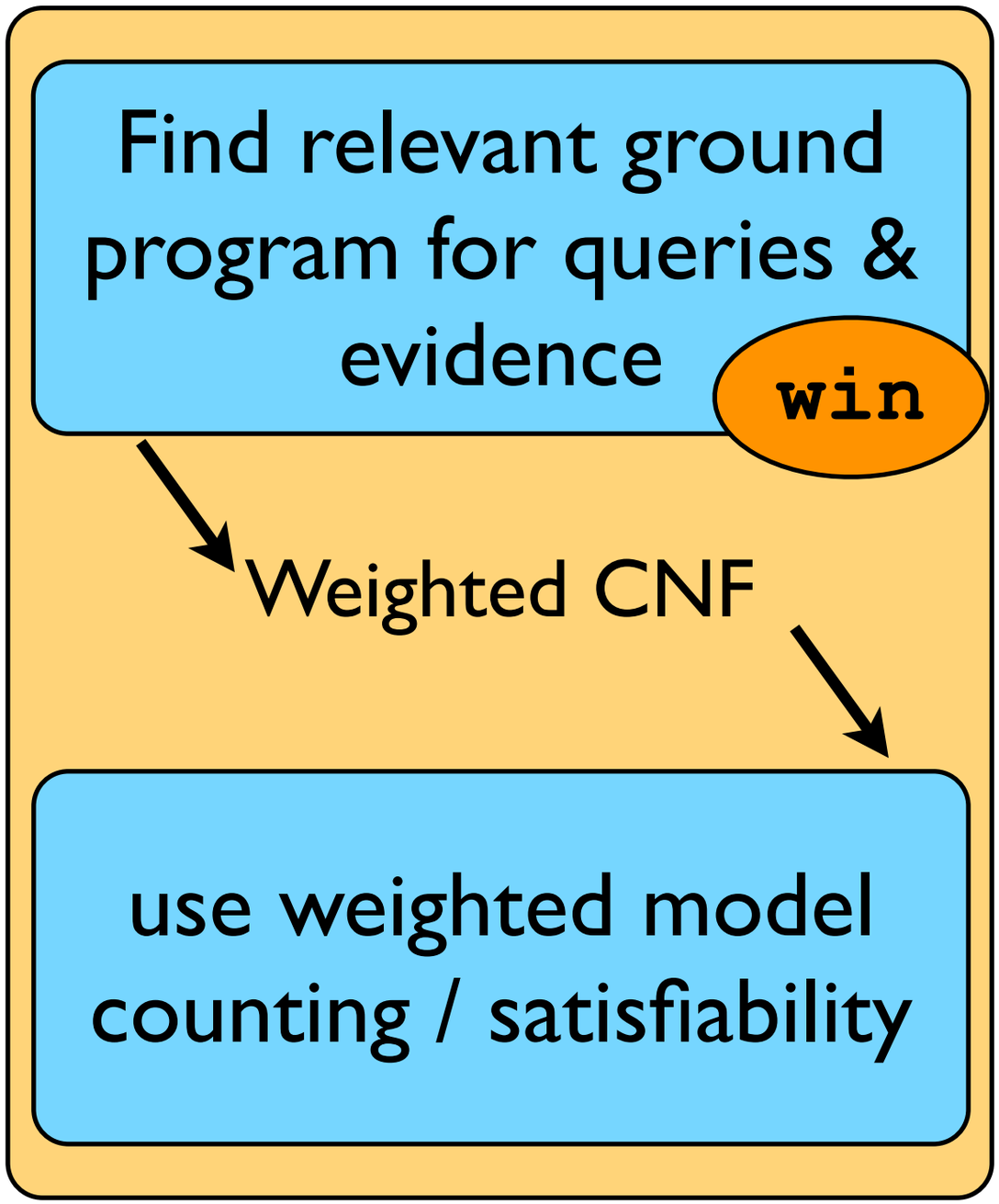


[Fierens et al, TPLP 14]

Current Approach

(ProbLog2)

```
0.4::heads(1).
0.7::heads(2).
0.5::heads(3).
win :- heads(1).
win :- heads(2),
       heads(3).
```



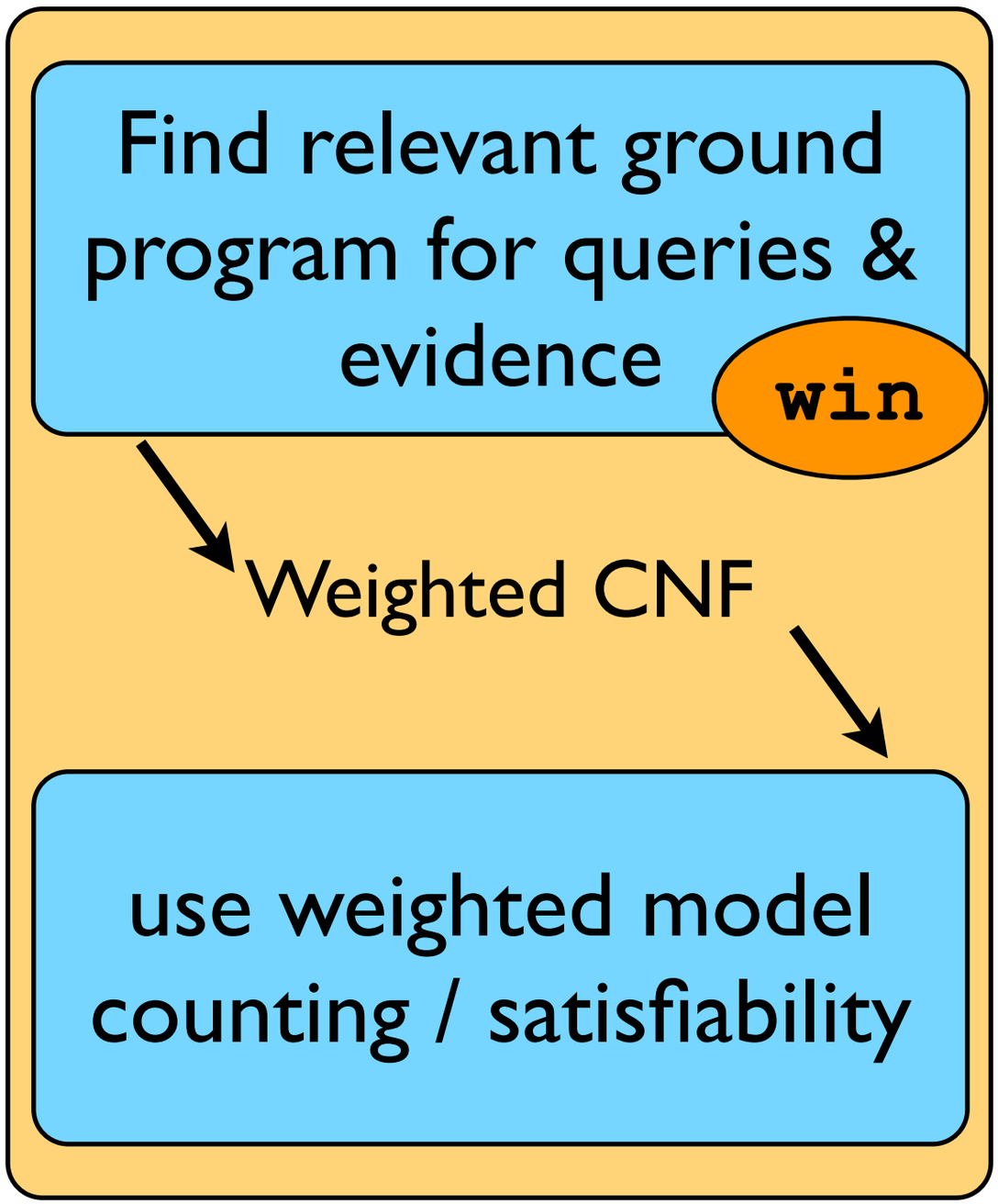

```
win :- heads(1).
win :- heads(2), heads(3).
```


[Fierens et al, TPLP 14]

Current Approach

(ProbLog2)

```
0.4::heads(1).
0.7::heads(2).
0.5::heads(3).
win :- heads(1).
win :- heads(2),
       heads(3).
```




```
win :- heads(1).
win :- heads(2), heads(3).
```

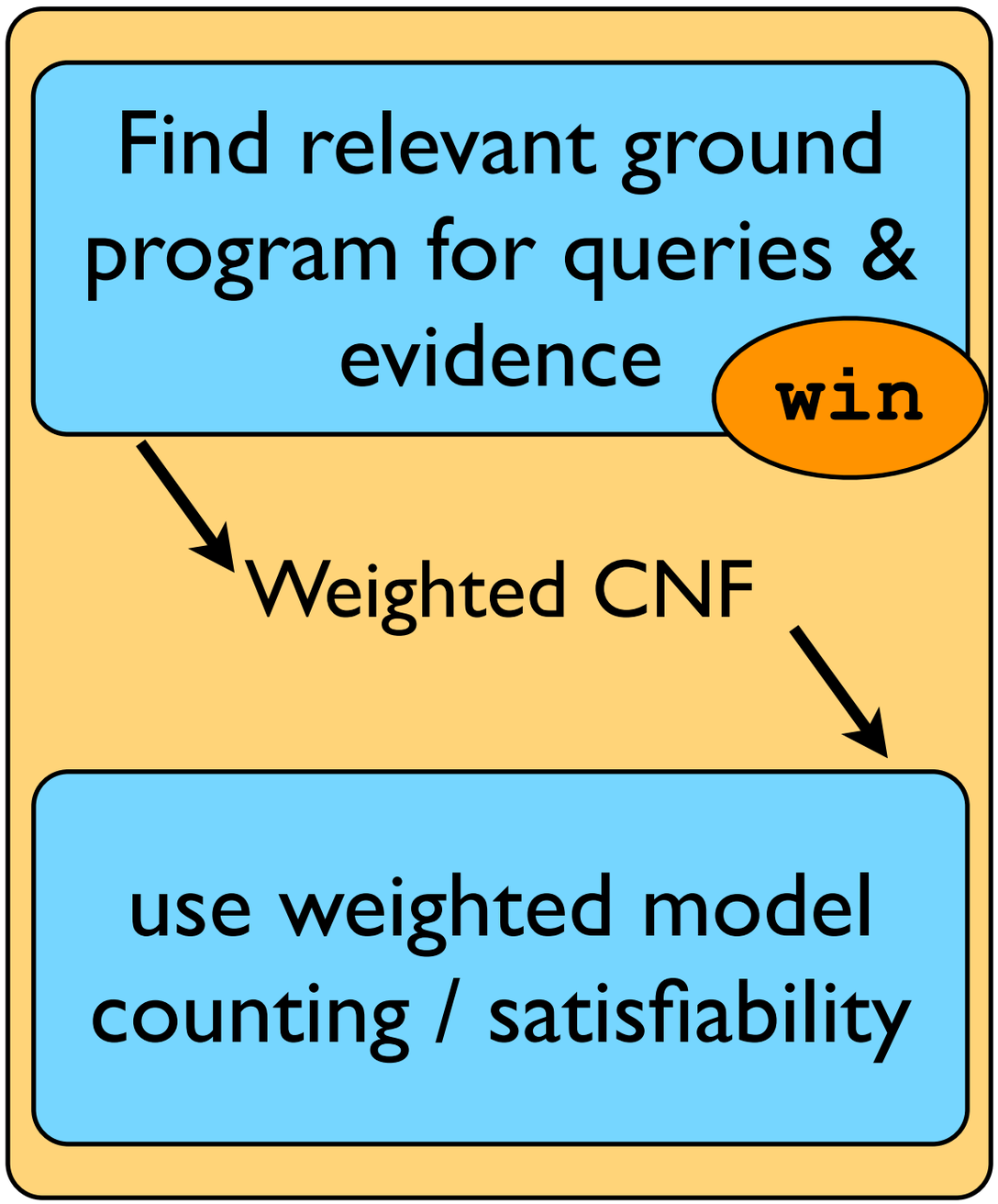
$$\text{win} \leftrightarrow h(1) \vee (h(2) \wedge h(3))$$

[Fierens et al, TPLP 14]

Current Approach

(ProbLog2)

```
0.4::heads(1).
0.7::heads(2).
0.5::heads(3).
win :- heads(1).
win :- heads(2),
       heads(3).
```




```
win :- heads(1).
win :- heads(2), heads(3).
```

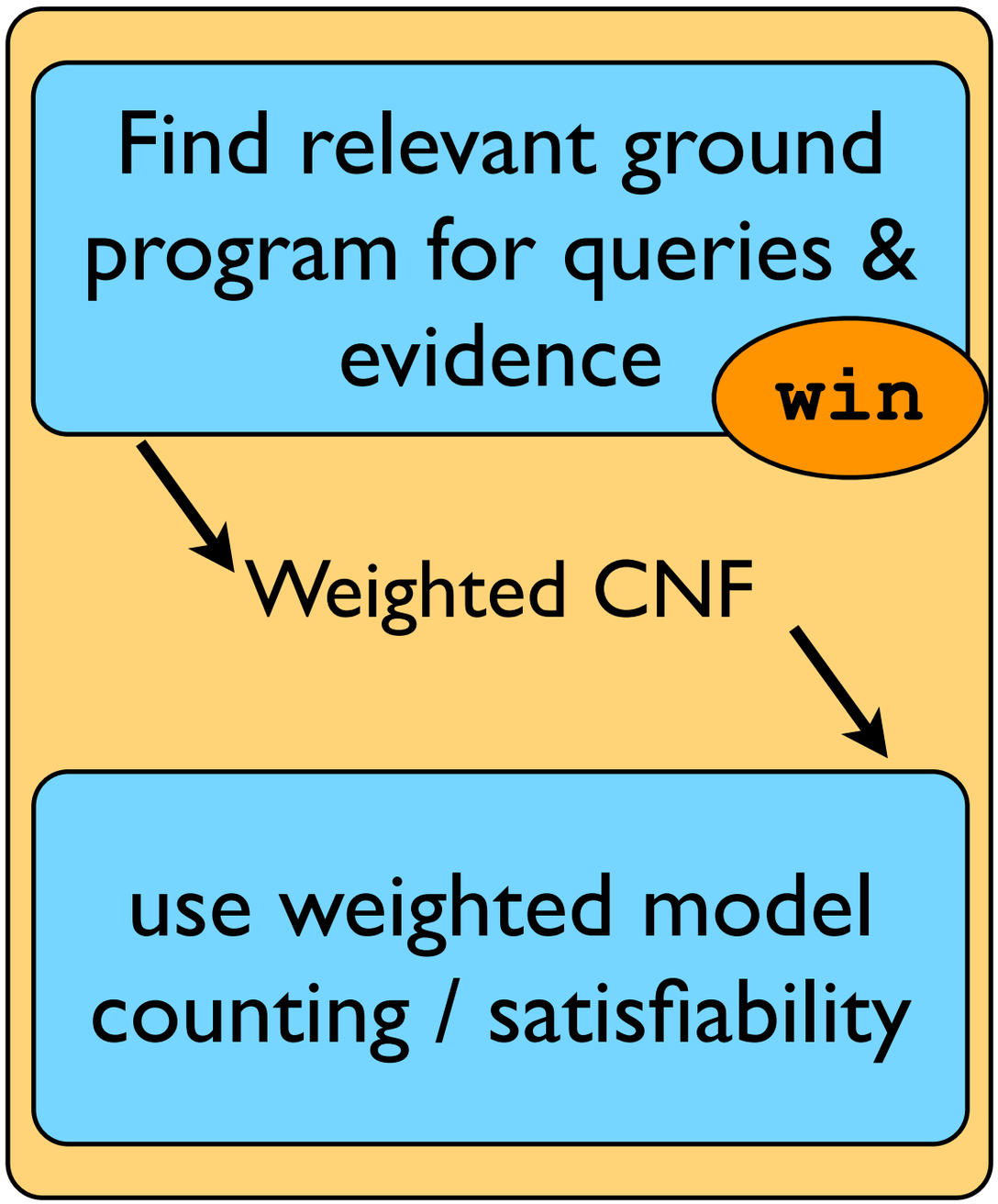
$$\text{win} \leftrightarrow h(1) \vee (h(2) \wedge h(3))$$

$$(\neg\text{win} \vee h(1) \vee h(2)) \wedge (\neg\text{win} \vee h(1) \vee h(3)) \wedge (\text{win} \vee \neg h(1)) \wedge (\text{win} \vee \neg h(2) \vee \neg h(3))$$

[Fierens et al, TPLP 14]

Current Approach
(ProbLog2)

```
0.4::heads(1).
0.7::heads(2).
0.5::heads(3).
win :- heads(1).
win :- heads(2),
       heads(3).
```




```
win :- heads(1).
win :- heads(2), heads(3).
```

$$\text{win} \leftrightarrow h(1) \vee (h(2) \wedge h(3))$$

$$(\neg\text{win} \vee h(1) \vee h(2)) \wedge (\neg\text{win} \vee h(1) \vee h(3)) \wedge (\text{win} \vee \neg h(1)) \wedge (\text{win} \vee \neg h(2) \vee \neg h(3))$$

$h(1) \rightarrow 0.4$	$h(2) \rightarrow 0.7$	$h(3) \rightarrow 0.5$
$\neg h(1) \rightarrow 0.6$	$\neg h(2) \rightarrow 0.3$	$\neg h(3) \rightarrow 0.5$

[Fierens et al, TPLP 14]

Current Approach
(ProbLog2)

```
0.4::heads(1).
0.7::heads(2).
0.5::heads(3).
win :- heads(1).
win :- heads(2),
       heads(3).
```

Find relevant ground program for queries & evidence

win

Weighted CNF

use weighted model counting / satisfiability

```
win :- heads(1).
win :- heads(2), heads(3).
```

$$\text{win} \leftrightarrow h(1) \vee (h(2) \wedge h(3))$$

$$(\neg\text{win} \vee h(1) \vee h(2)) \wedge (\neg\text{win} \vee h(1) \vee h(3)) \wedge (\text{win} \vee \neg h(1)) \wedge (\text{win} \vee \neg h(2) \vee \neg h(3))$$

use standard tool

$h(1) \rightarrow 0.4$ $\quad h(2) \rightarrow 0.7$ $\quad h(3) \rightarrow 0.5$

$\neg h(1) \rightarrow 0.6$ $\quad \neg h(2) \rightarrow 0.3$ $\quad \neg h(3) \rightarrow 0.5$

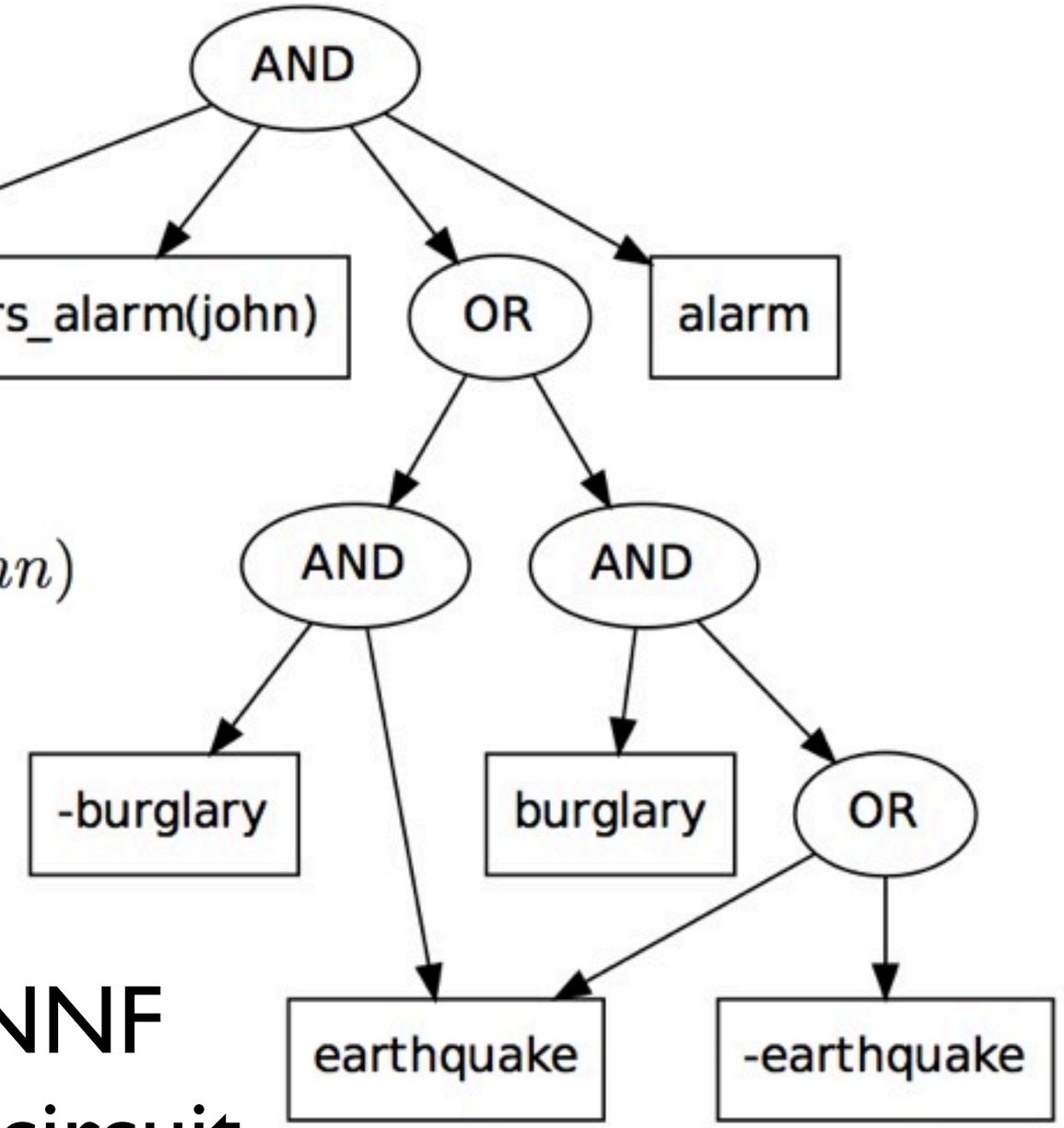
[Fierens et al, TPLP 14]

WMC using d-DNNFs

$$alarm \leftrightarrow burglary \vee earthquake$$

$$calls(john) \leftrightarrow alarm, hears\_alarm(john)$$

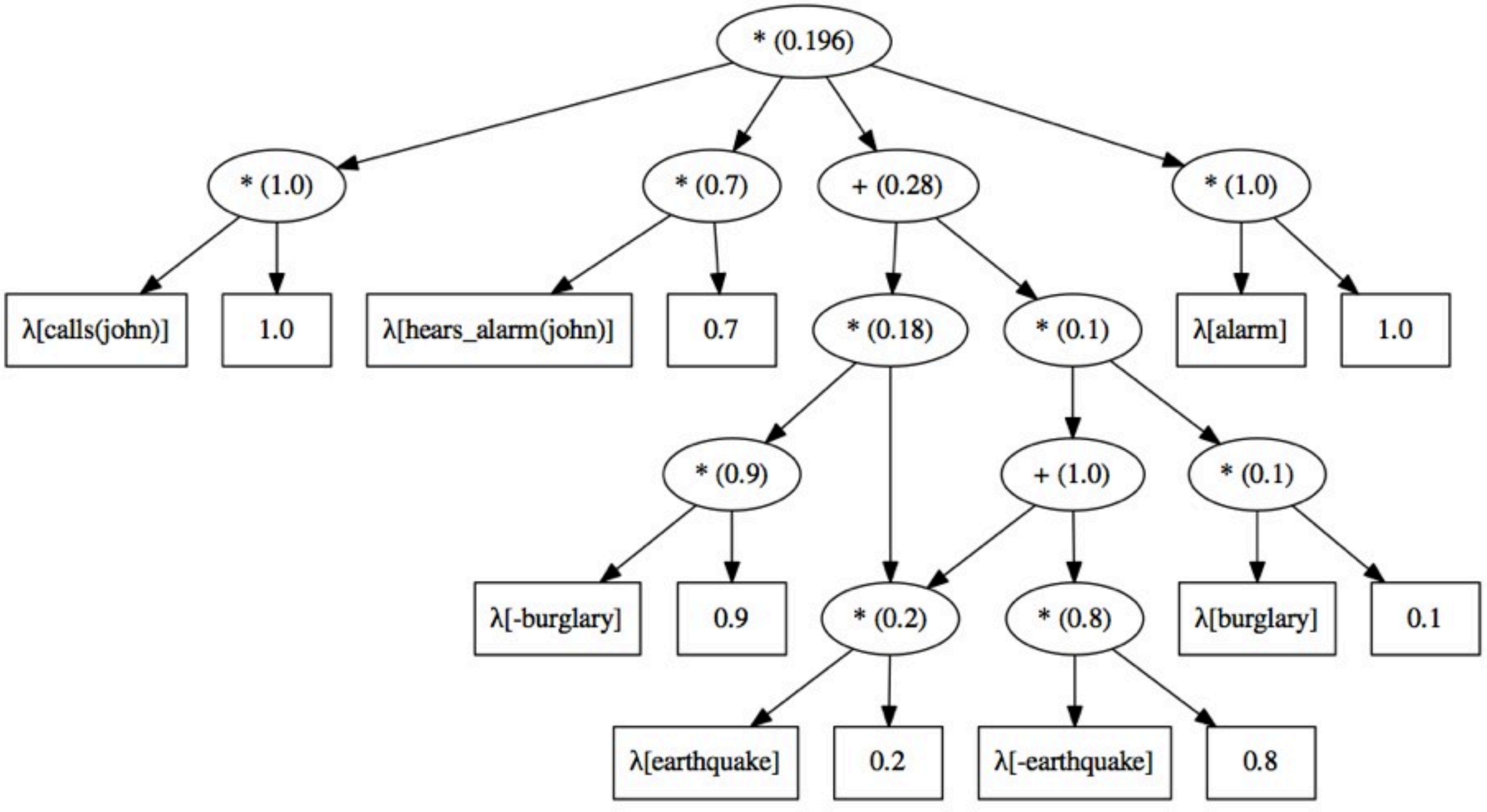
$$calls(john)$$



1. represent formula as d-DNNF
2. transform into arithmetic circuit
3. evaluate bottom-up

[Figure: Fierens et al, TPLP 14]

WMC using d-DNNFs



3. evaluate bottom-up

[Figure: Fierens et al, TPLP 14]

ProbLog Inference

- reduction to propositional formula
- addresses disjoint-sum-problem
- **but**: not all probabilistic logic programs face this problem! e.g., weather
- more generally: mutually exclusive proofs as assumed in PRISM

PRISM

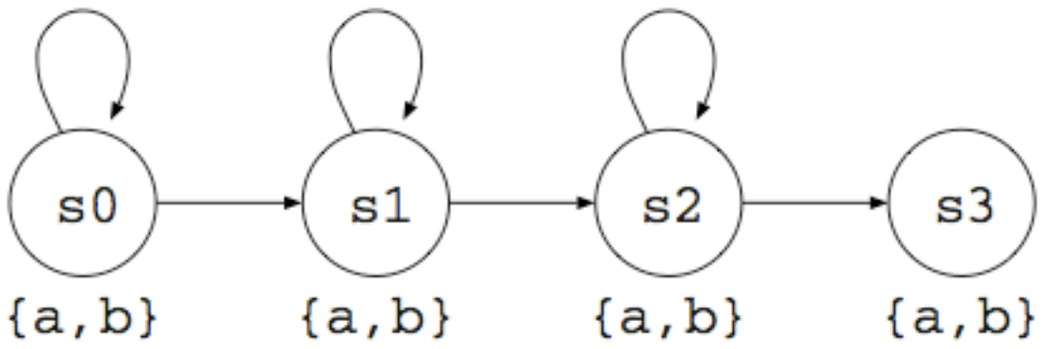


Fig. 1. Example of a left-to-right HMM with four states

```
target(hmm/1).
values(tr(s0),[s0,s1]).
values(tr(s1),[s1,s2]).
values(tr(s2),[s2,s3]).
values(out(_),[a,b]).

hmm(Cs):- hmm(0,s0,Cs).

hmm(T,s3,[C]):- msw(out(s3),  C). % If at the final state:
                                  %   output a symbol and then terminate.
hmm(T,S,[C|Cs]):- S\==s3,         % If not at the final state:
  msw(out(S),  C),                %   choose a symbol to be output,
  msw(tr(S),  Next),              %   choose the next state,
  T1 is T+1,                      %   Put the clock ahead,
  hmm(T1,Next,Cs).                %   and enter the next loop.
```

Fig. 2. PRISM program for the left-to-right HMM

[Figures: Sato and Kameya 08]

PRISM

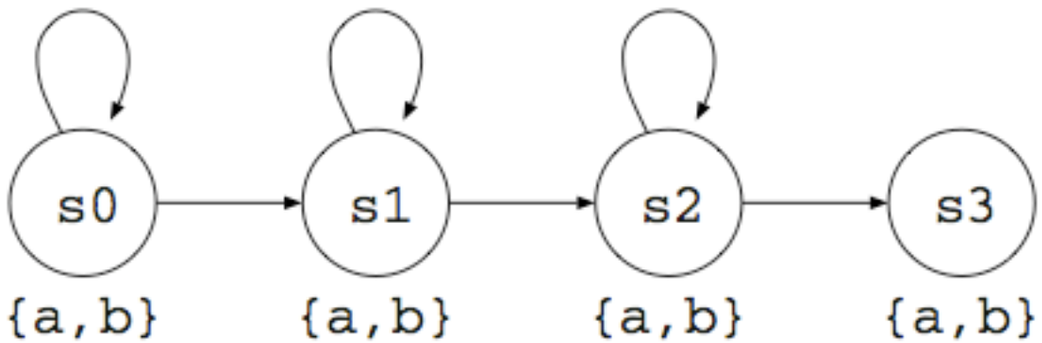


Fig. 1. Example of a left-to-right HMM with four states

```
target(hmm/1).
values(tr(s0),[s0,s1]).
values(tr(s1),[s1,s2]).
values(tr(s2),[s2,s3]).
values(out(_),[a,b]).
```

define switches

```
hmm(Cs):- hmm(0,s0,Cs).

hmm(T,s3,[C]):- msw(out(s3),  C). % If at the final state:
                                  %   output a symbol and then terminate.
hmm(T,S,[C|Cs]):- S\==s3,         % If not at the final state:
  msw(out(S),  C),                %   choose a symbol to be output,
  msw(tr(S),  Next),              %   choose the next state,
  T1 is T+1,                      %   Put the clock ahead,
  hmm(T1,Next,Cs).                %   and enter the next loop.
```

Fig. 2. PRISM program for the left-to-right HMM

[Figures: Sato and Kameya 08]

PRISM

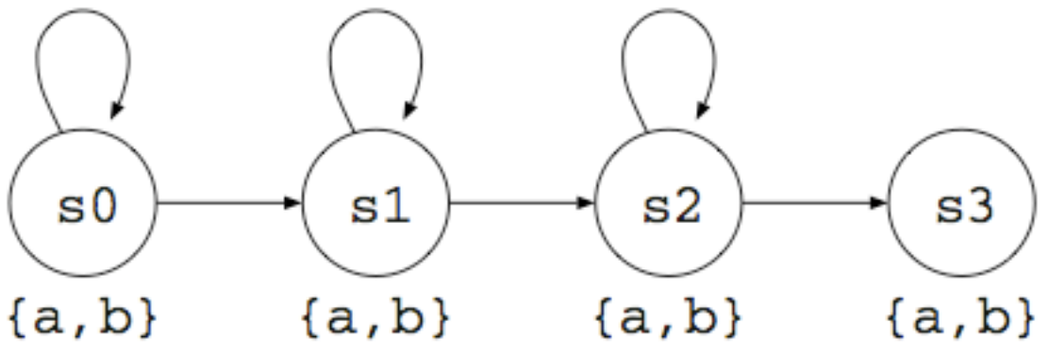


Fig. 1. Example of a left-to-right HMM with four states

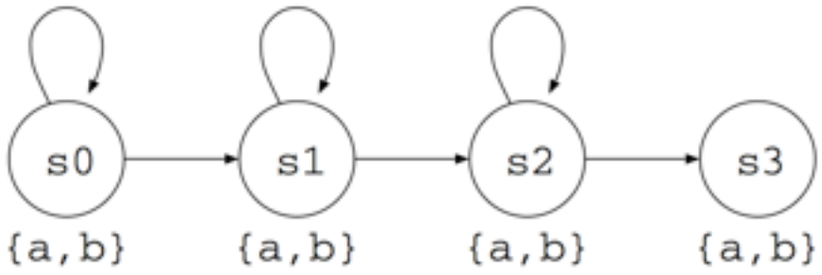
```
target(hmm/1).
values(tr(s0),[s0,s1]).
values(tr(s1),[s1,s2]).
values(tr(s2),[s2,s3]).
values(out(_),[a,b]).

hmm(Cs):- hmm(0,s0,Cs).

hmm(T,s3,[C]):- msw(out(s3), C). % If at the final state:
                                 %   output a symbol and then terminate.
hmm(T,S,[C|Cs]):- S\==s3,        % If not at the final state:
  msw(out(S), C),                %   choose a symbol to be output,
  msw(tr(S), Next),              %   choose the next state,
  T1 is T+1,                     %   Put the clock ahead,
  hmm(T1,Next,Cs).               %   and enter the next loop.
```

use switches

Fig. 2. PRISM program for the left-to-right HMM

[Figures: Sato and Kameya 08]

PRISM inference




```
hmm([a,b,b,b,b,a])
```

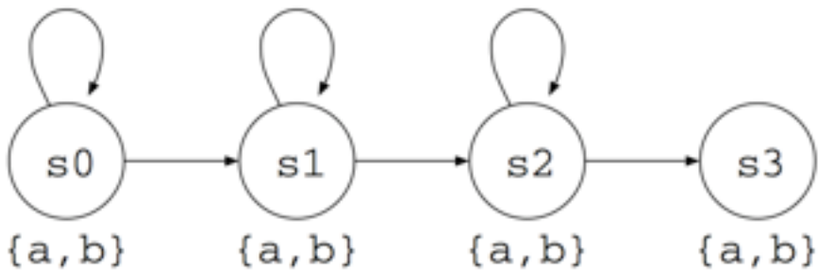
no memoization: two different RVS

$$
\begin{aligned}
E_1 = {} & \mathtt{m(out(s0),a)} \wedge \mathtt{m(tr(s0),s0)} \wedge \mathtt{m(out(s0),b)} \wedge \mathtt{m(tr(s0),s0)} \wedge \mathtt{m(out(s0),b)} \\
& \wedge \mathtt{m(tr(s0),s1)} \wedge \mathtt{m(out(s1),b)} \wedge \mathtt{m(tr(s1),s2)} \wedge \mathtt{m(out(s2),b)} \wedge \mathtt{m(tr(s2),s3)} \\
& \wedge \mathtt{m(out(s3),a)} \\
E_2 = {} & \mathtt{m(out(s0),a)} \wedge \mathtt{m(tr(s0),s0)} \wedge \mathtt{m(out(s0),b)} \wedge \mathtt{m(tr(s0),s1)} \wedge \mathtt{m(out(s1),b)} \\
& \wedge \mathtt{m(tr(s1),s1)} \wedge \mathtt{m(out(s1),b)} \wedge \mathtt{m(tr(s1),s2)} \wedge \mathtt{m(out(s2),b)} \wedge \mathtt{m(tr(s2),s3)} \\
& \wedge \mathtt{m(out(s3),a)} \\
& \vdots \\
E_6 = {} & \mathtt{m(out(s0),a)} \wedge \mathtt{m(tr(s0),s1)} \wedge \mathtt{m(out(s1),b)} \wedge \mathtt{m(tr(s1),s2)} \wedge \mathtt{m(out(s2),b)} \\
& \wedge \mathtt{m(tr(s2),s2)} \wedge \mathtt{m(out(s2),b)} \wedge \mathtt{m(tr(s2),s2)} \wedge \mathtt{m(out(s2),b)} \wedge \mathtt{m(tr(s2),s3)} \\
& \wedge \mathtt{m(out(s3),a)}
\end{aligned}
$$

Fig. 3. Six explanations for `hmm([a, b, b, b, b, a])`. Due to the space limit, the predicate name `msw` is abbreviated to `m`.

[Figures: Sato and Kameya 08]

PRISM inference

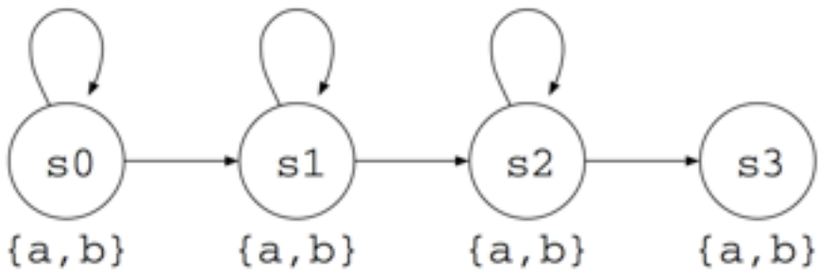


$$
\begin{aligned}
\texttt{hmm}([\texttt{a},\texttt{b},\texttt{b},\texttt{b},\texttt{b},\texttt{a}]) &\Leftrightarrow \texttt{hmm}(0,\texttt{s0},[\texttt{a},\texttt{b},\texttt{b},\texttt{b},\texttt{b},\texttt{a}])\\
\texttt{hmm}(0,\texttt{s0},[\texttt{a},\texttt{b},\texttt{b},\texttt{b},\texttt{b},\texttt{a}]) &\Leftrightarrow \texttt{m}(\texttt{out}(\texttt{s0}),\texttt{a}) \wedge \texttt{m}(\texttt{tr}(\texttt{s0}),\texttt{s0}) \wedge \texttt{hmm}(1,\texttt{s0},[\texttt{b},\texttt{b},\texttt{b},\texttt{b},\texttt{a}])\\
&\quad \vee \texttt{m}(\texttt{out}(\texttt{s0}),\texttt{a}) \wedge \texttt{m}(\texttt{tr}(\texttt{s0}),\texttt{s1}) \wedge \texttt{hmm}(1,\texttt{s1},[\texttt{b},\texttt{b},\texttt{b},\texttt{b},\texttt{a}])\\
\texttt{hmm}(1,\texttt{s0},[\texttt{b},\texttt{b},\texttt{b},\texttt{b},\texttt{a}]) &\Leftrightarrow \texttt{m}(\texttt{out}(\texttt{s0}),\texttt{b}) \wedge \texttt{m}(\texttt{tr}(\texttt{s0}),\texttt{s0}) \wedge \texttt{hmm}(2,\texttt{s0},[\texttt{b},\texttt{b},\texttt{b},\texttt{a}])\\
&\quad \vee \texttt{m}(\texttt{out}(\texttt{s0}),\texttt{b}) \wedge \texttt{m}(\texttt{tr}(\texttt{s0}),\texttt{s1}) \wedge \texttt{hmm}(2,\texttt{s1},[\texttt{b},\texttt{b},\texttt{b},\texttt{a}])\ddagger\\
\texttt{hmm}(2,\texttt{s0},[\texttt{b},\texttt{b},\texttt{b},\texttt{a}]) &\Leftrightarrow \texttt{m}(\texttt{out}(\texttt{s0}),\texttt{b}) \wedge \texttt{m}(\texttt{tr}(\texttt{s0}),\texttt{s1}) \wedge \texttt{hmm}(3,\texttt{s1},[\texttt{b},\texttt{b},\texttt{a}])\\
\texttt{hmm}(1,\texttt{s1},[\texttt{b},\texttt{b},\texttt{b},\texttt{b},\texttt{a}]) &\Leftrightarrow \texttt{m}(\texttt{out}(\texttt{s1}),\texttt{b}) \wedge \texttt{m}(\texttt{tr}(\texttt{s1}),\texttt{s1}) \wedge \texttt{hmm}(2,\texttt{s1},[\texttt{b},\texttt{b},\texttt{b},\texttt{a}])\ddagger\\
&\quad \vee \texttt{m}(\texttt{out}(\texttt{s1}),\texttt{b}) \wedge \texttt{m}(\texttt{tr}(\texttt{s1}),\texttt{s2}) \wedge \texttt{hmm}(2,\texttt{s2},[\texttt{b},\texttt{b},\texttt{b},\texttt{a}])\\
\texttt{hmm}(2,\texttt{s1},[\texttt{b},\texttt{b},\texttt{b},\texttt{a}])\dagger &\Leftrightarrow \texttt{m}(\texttt{out}(\texttt{s1}),\texttt{b}) \wedge \texttt{m}(\texttt{tr}(\texttt{s1}),\texttt{s1}) \wedge \texttt{hmm}(3,\texttt{s1},[\texttt{b},\texttt{b},\texttt{a}])\\
&\quad \vee \texttt{m}(\texttt{out}(\texttt{s1}),\texttt{b}) \wedge \texttt{m}(\texttt{tr}(\texttt{s1}),\texttt{s2}) \wedge \texttt{hmm}(3,\texttt{s2},[\texttt{b},\texttt{b},\texttt{a}])\\
\texttt{hmm}(3,\texttt{s1},[\texttt{b},\texttt{b},\texttt{a}]) &\Leftrightarrow \texttt{m}(\texttt{out}(\texttt{s1}),\texttt{b}) \wedge \texttt{m}(\texttt{tr}(\texttt{s1}),\texttt{s2}) \wedge \texttt{hmm}(4,\texttt{s2},[\texttt{b},\texttt{a}])\\
\texttt{hmm}(2,\texttt{s2},[\texttt{b},\texttt{b},\texttt{b},\texttt{a}]) &\Leftrightarrow \texttt{m}(\texttt{out}(\texttt{s2}),\texttt{b}) \wedge \texttt{m}(\texttt{tr}(\texttt{s2}),\texttt{s2}) \wedge \texttt{hmm}(3,\texttt{s2},[\texttt{b},\texttt{b},\texttt{a}])\\
\texttt{hmm}(3,\texttt{s2},[\texttt{b},\texttt{b},\texttt{a}]) &\Leftrightarrow \texttt{m}(\texttt{out}(\texttt{s2}),\texttt{b}) \wedge \texttt{m}(\texttt{tr}(\texttt{s2}),\texttt{s2}) \wedge \texttt{hmm}(4,\texttt{s2},[\texttt{b},\texttt{a}])\\
\texttt{hmm}(4,\texttt{s2},[\texttt{b},\texttt{a}]) &\Leftrightarrow \texttt{m}(\texttt{out}(\texttt{s2}),\texttt{b}) \wedge \texttt{m}(\texttt{tr}(\texttt{s2}),\texttt{s3}) \wedge \texttt{hmm}(5,\texttt{s3},[\texttt{a}])\\
\texttt{hmm}(5,\texttt{s3},[\texttt{a}]) &\Leftrightarrow \texttt{m}(\texttt{out}(\texttt{s3}),\texttt{a})
\end{aligned}
$$

Fig. 4. Factorized explanations for $\texttt{hmm}([\texttt{a},\texttt{b},\texttt{b},\texttt{b},\texttt{b},\texttt{a}])$

[Figures: Sato and Kameya 08]

PRISM inference



$$
\begin{aligned}
\texttt{hmm}([\texttt{a},\texttt{b},\texttt{b},\texttt{b},\texttt{b},\texttt{a}]) &\Leftrightarrow \texttt{hmm}(0,\texttt{s0},[\texttt{a},\texttt{b},\texttt{b},\texttt{b},\texttt{b},\texttt{a}])\\
\texttt{hmm}(0,\texttt{s0},[\texttt{a},\texttt{b},\texttt{b},\texttt{b},\texttt{b},\texttt{a}]) &\Leftrightarrow \texttt{m}(\texttt{out}(\texttt{s0}),\texttt{a}) \wedge \texttt{m}(\texttt{tr}(\texttt{s0}),\texttt{s0}) \wedge \texttt{hmm}(1,\texttt{s0},[\texttt{b},\texttt{b},\texttt{b},\texttt{b},\texttt{a}])\\
&\quad \vee \texttt{m}(\texttt{out}(\texttt{s0}),\texttt{a}) \wedge \texttt{m}(\texttt{tr}(\texttt{s0}),\texttt{s1}) \wedge \texttt{hmm}(1,\texttt{s1},[\texttt{b},\texttt{b},\texttt{b},\texttt{b},\texttt{a}])\\
\texttt{hmm}(1,\texttt{s0},[\texttt{b},\texttt{b},\texttt{b},\texttt{b},\texttt{a}]) &\Leftrightarrow \texttt{m}(\texttt{out}(\texttt{s0}),\texttt{b}) \wedge \texttt{m}(\texttt{tr}(\texttt{s0}),\texttt{s0}) \wedge \texttt{hmm}(2,\texttt{s0},[\texttt{b},\texttt{b},\texttt{b},\texttt{a}])\\
&\quad \vee \texttt{m}(\texttt{out}(\texttt{s0}),\texttt{b}) \wedge \texttt{m}(\texttt{tr}(\texttt{s0}),\texttt{s1}) \wedge \texttt{hmm}(2,\texttt{s1},[\texttt{b},\texttt{b},\texttt{b},\texttt{a}])\ddagger\\
\texttt{hmm}(2,\texttt{s0},[\texttt{b},\texttt{b},\texttt{b},\texttt{a}]) &\Leftrightarrow \texttt{m}(\texttt{out}(\texttt{s0}),\texttt{b}) \wedge \texttt{m}(\texttt{tr}(\texttt{s0}),\texttt{s1}) \wedge \texttt{hmm}(3,\texttt{s1},[\texttt{b},\texttt{b},\texttt{a}])\\
\texttt{hmm}(1,\texttt{s1},[\texttt{b},\texttt{b},\texttt{b},\texttt{b},\texttt{a}]) &\Leftrightarrow \texttt{m}(\texttt{out}(\texttt{s1}),\texttt{b}) \wedge \texttt{m}(\texttt{tr}(\texttt{s1}),\texttt{s1}) \wedge \texttt{hmm}(2,\texttt{s1},[\texttt{b},\texttt{b},\texttt{b},\texttt{a}])\ddagger\\
&\quad \vee \texttt{m}(\texttt{out}(\texttt{s1}),\texttt{b}) \wedge \texttt{m}(\texttt{tr}(\texttt{s1}),\texttt{s2}) \wedge \texttt{hmm}(2,\texttt{s2},[\texttt{b},\texttt{b},\texttt{b},\texttt{a}])\\
\texttt{hmm}(2,\texttt{s1},[\texttt{b},\texttt{b},\texttt{b},\texttt{a}])\dagger &\Leftrightarrow \texttt{m}(\texttt{out}(\texttt{s1}),\texttt{b}) \wedge \texttt{m}(\texttt{tr}(\texttt{s1}),\texttt{s1}) \wedge \texttt{hmm}(3,\texttt{s1},[\texttt{b},\texttt{b},\texttt{a}])\\
&\quad \vee \texttt{m}(\texttt{out}(\texttt{s1}),\texttt{b}) \wedge \texttt{m}(\texttt{tr}(\texttt{s1}),\texttt{s2}) \wedge \texttt{hmm}(3,\texttt{s2},[\texttt{b},\texttt{b},\texttt{a}])\\
\texttt{hmm}(3,\texttt{s1},[\texttt{b},\texttt{b},\texttt{a}]) &\Leftrightarrow \texttt{m}(\texttt{out}(\texttt{s1}),\texttt{b}) \wedge \texttt{m}(\texttt{tr}(\texttt{s1}),\texttt{s2}) \wedge \texttt{hmm}(4,\texttt{s2},[\texttt{b},\texttt{a}])\\
\texttt{hmm}(2,\texttt{s2},[\texttt{b},\texttt{b},\texttt{b},\texttt{a}]) &\Leftrightarrow \texttt{m}(\texttt{out}(\texttt{s2}),\texttt{b}) \wedge \texttt{m}(\texttt{tr}(\texttt{s2}),\texttt{s2}) \wedge \texttt{hmm}(3,\texttt{s2},[\texttt{b},\texttt{b},\texttt{a}])\\
\texttt{hmm}(3,\texttt{s2},[\texttt{b},\texttt{b},\texttt{a}]) &\Leftrightarrow \texttt{m}(\texttt{out}(\texttt{s2}),\texttt{b}) \wedge \texttt{m}(\texttt{tr}(\texttt{s2}),\texttt{s2}) \wedge \texttt{hmm}(4,\texttt{s2},[\texttt{b},\texttt{a}])\\
\texttt{hmm}(4,\texttt{s2},[\texttt{b},\texttt{a}]) &\Leftrightarrow \texttt{m}(\texttt{out}(\texttt{s2}),\texttt{b}) \wedge \texttt{m}(\texttt{tr}(\texttt{s2}),\texttt{s3}) \wedge \texttt{hmm}(5,\texttt{s3},[\texttt{a}])\\
\texttt{hmm}(5,\texttt{s3},[\texttt{a}]) &\Leftrightarrow \texttt{m}(\texttt{out}(\texttt{s3}),\texttt{a})
\end{aligned}
$$

Fig. 4. Factorized explanations for $\texttt{hmm}([\texttt{a},\texttt{b},\texttt{b},\texttt{b},\texttt{b},\texttt{a}])$

[Figures: Sato and Kameya 08]

PRISM: compute probability by dynamic programming

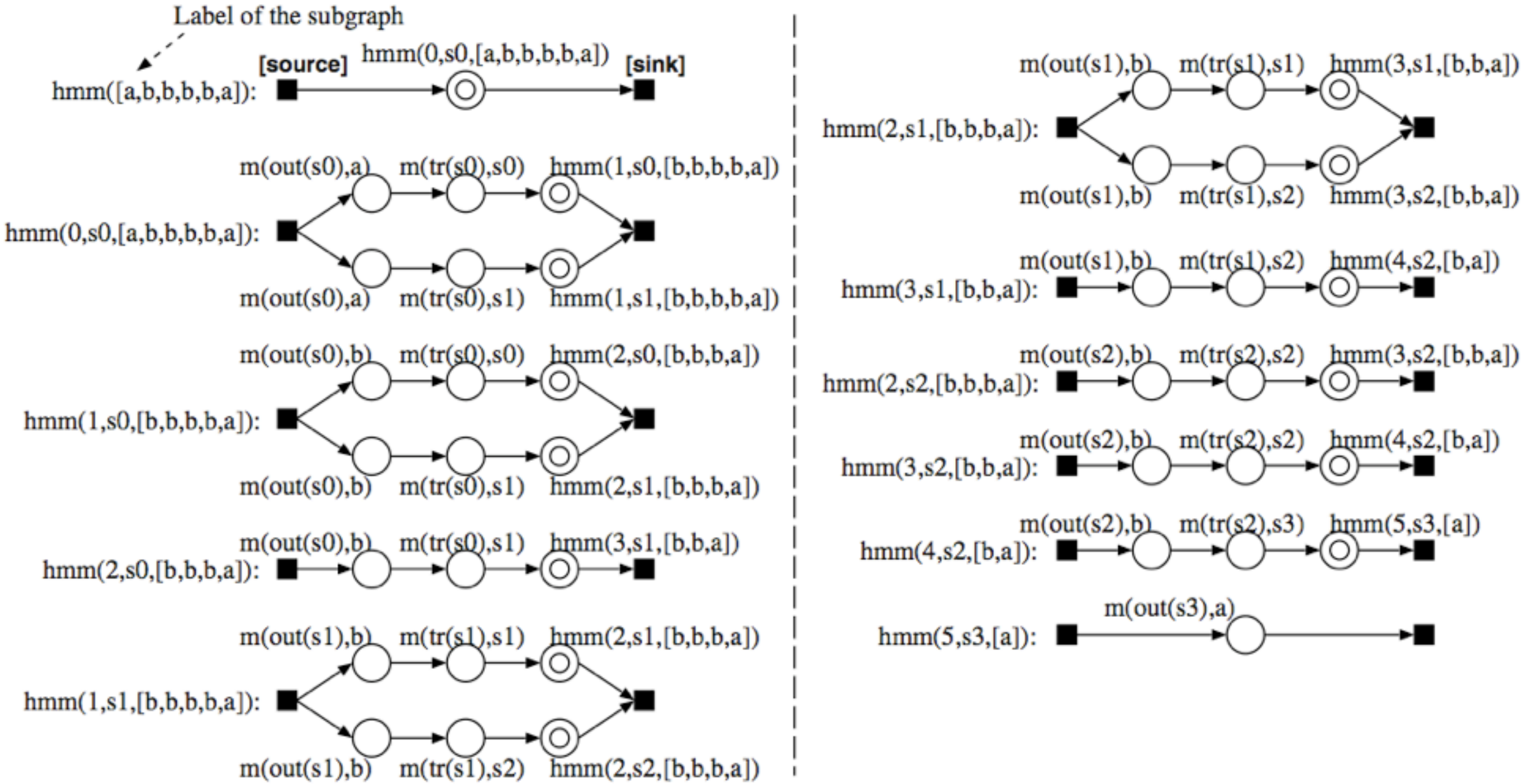


Fig. 5. Explanation graph

[Figures: Sato and Kameya 08]

PRISM: compute probability by dynamic programming

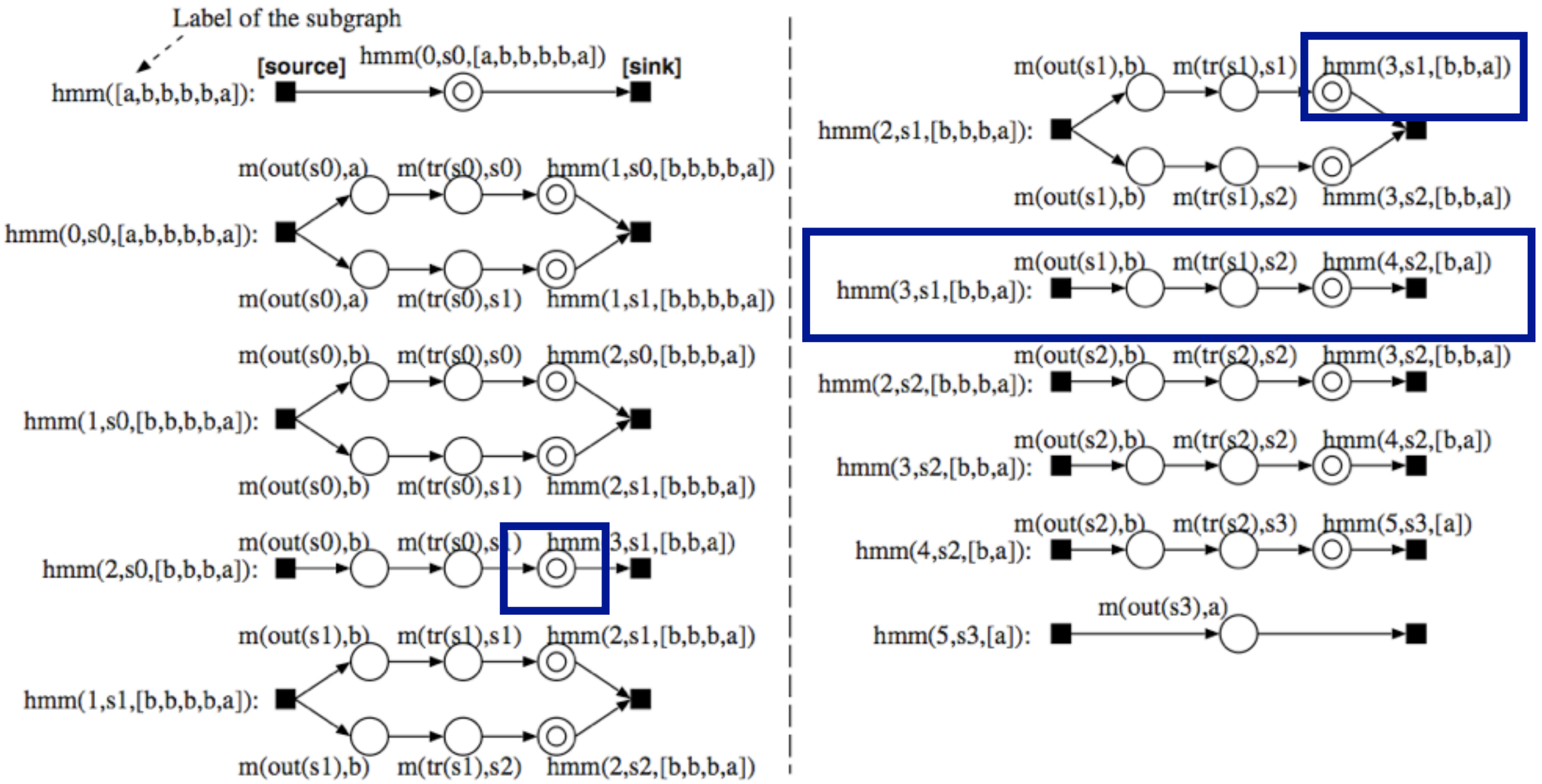


Fig. 5. Explanation graph

[Figures: Sato and Kameya 08]

Query Evaluation in PDB

- **Extensional** evaluation
  - guided by query expression only
  - exploit DB technology
  - for queries known to have polytime evaluation

Query Evaluation in PDB

- **Extensional** evaluation
  - guided by query expression only
  - exploit DB technology
  - for queries known to have polytime evaluation
- **Intensional** evaluation
  - construct **lineage** (= propositional formula)
  - compute probability of lineage
  - all queries

same idea as for ProbLog

Approximate Inference

- Lower and upper bounds

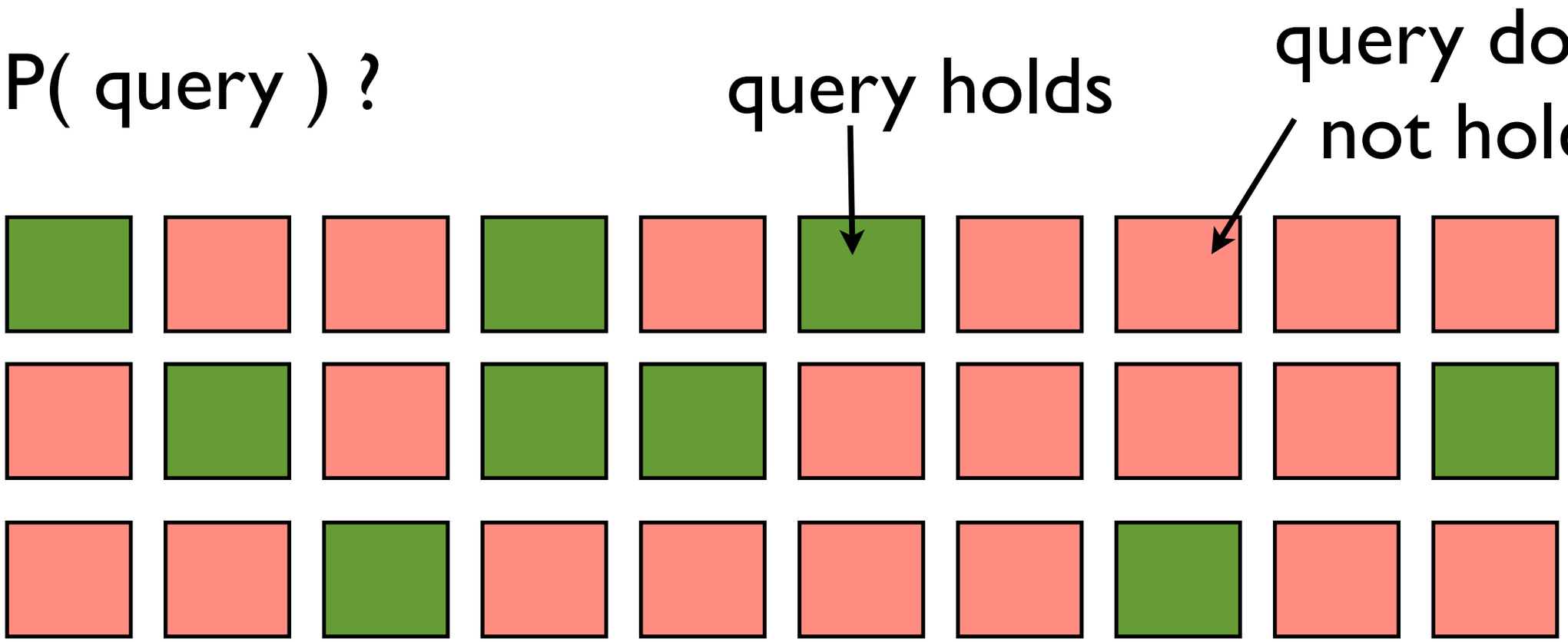
$$\phi_L \models \phi \models \phi_U$$

$$P(\phi_L) \leq P(\phi) \leq P(\phi_U)$$

- Sampling

Sampling

- P(query) ?



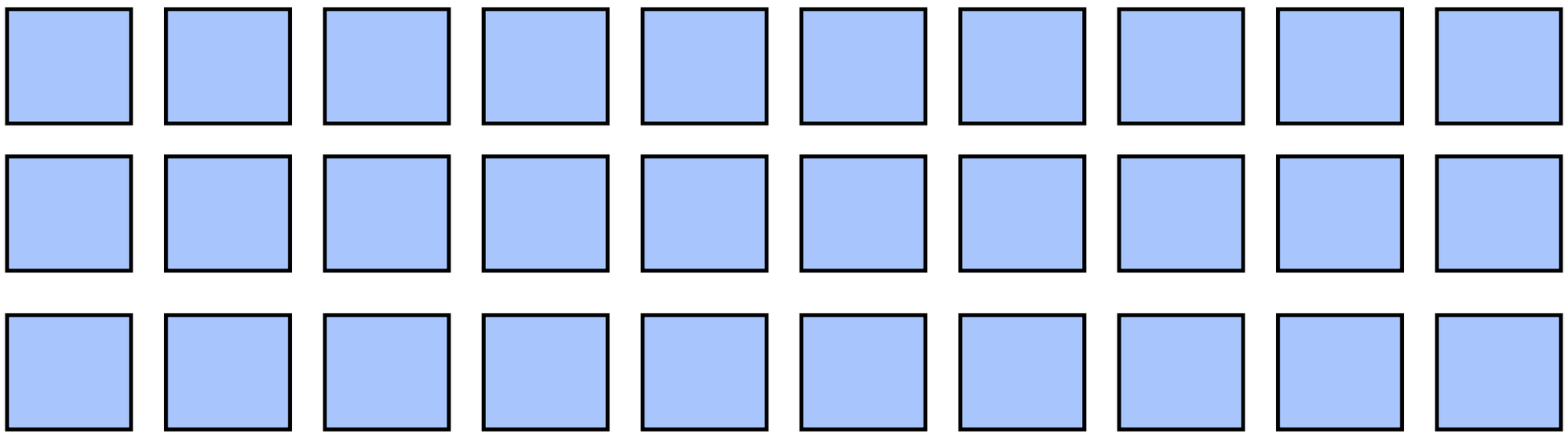
$$P(\text{query}) \approx \frac{\#\text{ query holds}}{\#\text{ worlds sampled}}$$

Rejection Sampling

- P(query | evidence) ?

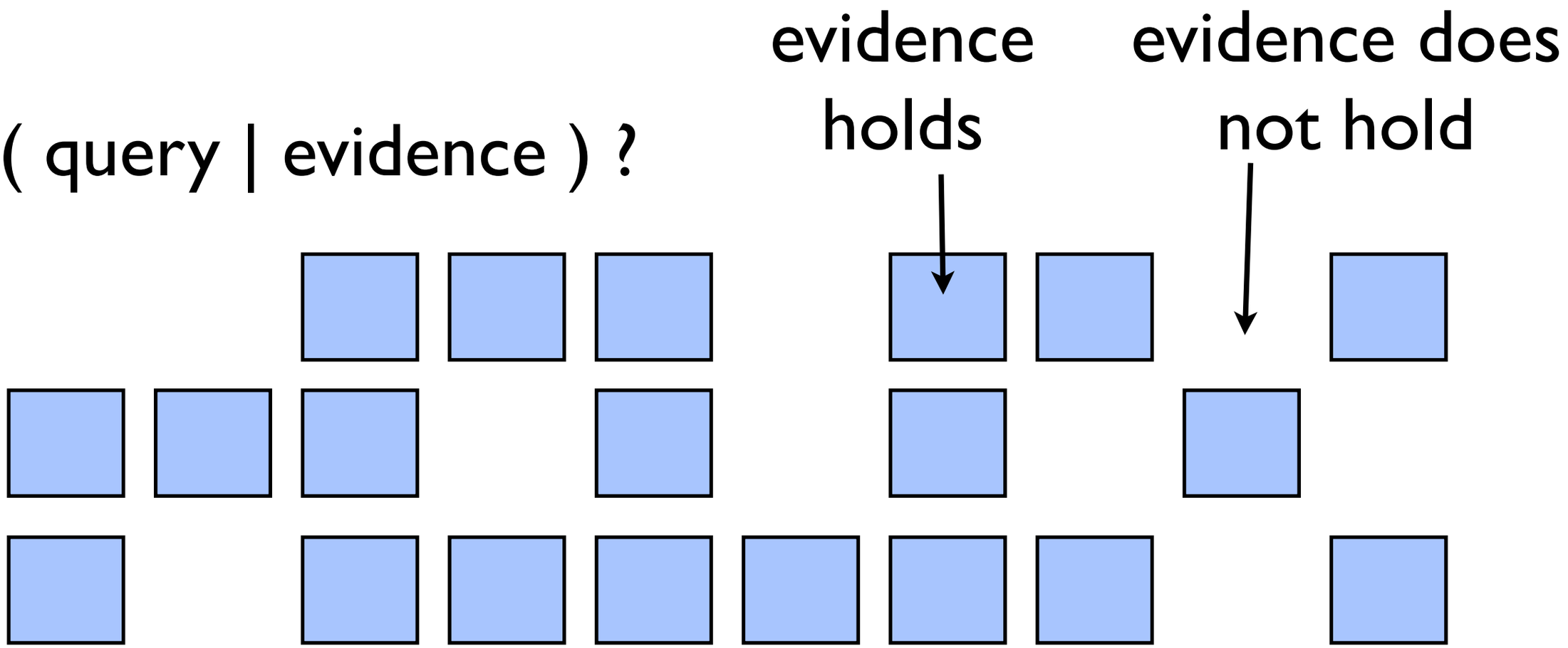
Rejection Sampling

- P(query | evidence) ?

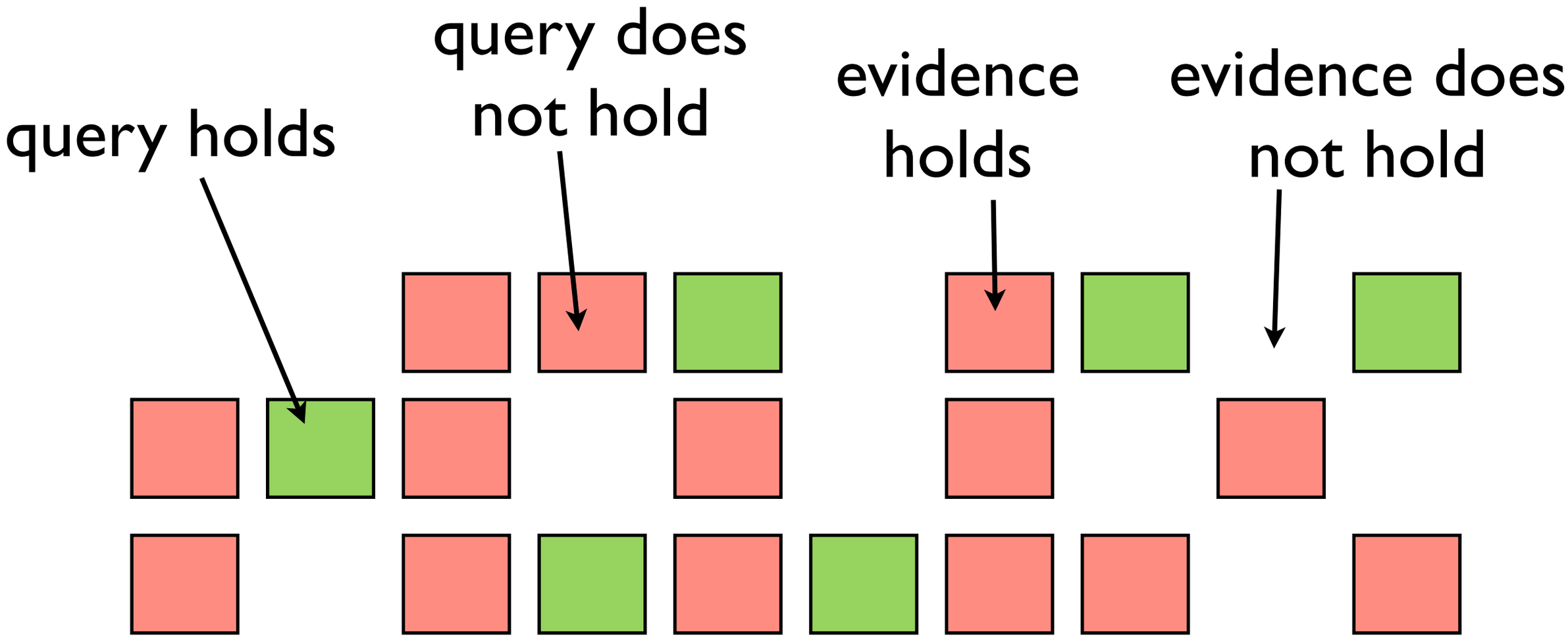


Rejection Sampling

- P(query | evidence) ?



Rejection Sampling



$$P(\text{query} \mid \text{evidence}) \approx \frac{\#\ \text{query \& evidence holds}}{\#\ \text{evidence holds}}$$

Markov Chain Monte Carlo (MCMC)

- Generate next sample by modifying current one
- Most common inference approach for PP languages such as Church, BLOG, ...
- Also considered for PRISM and ProbLog

Key challenges:
- how to propose next sample
- how to handle evidence

Parameter Learning

Parameter Learning

e.g., webpage classification model

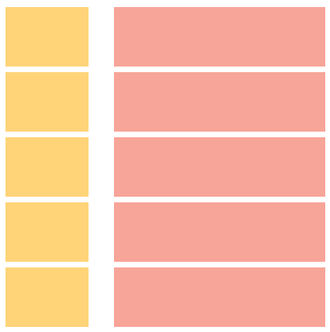
for each ***CLASS1***, ***CLASS2*** and each ***WORD***

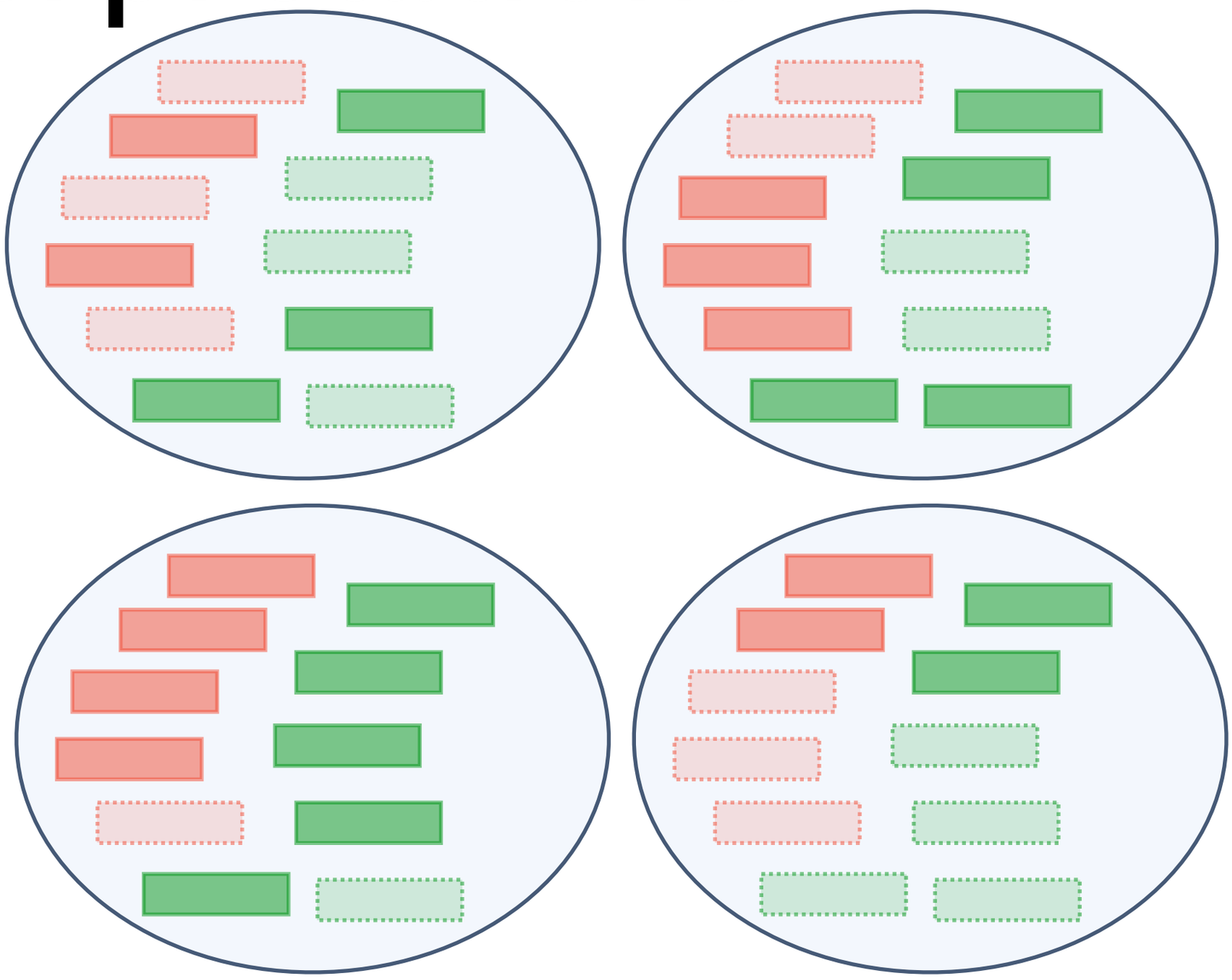
?? :: link_class(Source,Target,***CLASS1***,***CLASS2***).
?? :: word_class(***WORD***,***CLASS***).

```
class(Page,C)    :-  has_word(Page,W), word_class(W,C).

class(Page,C)    :-  links_to(OtherPage,Page),
                     class(OtherPage,OtherClass),
                     link_class(OtherPage,Page,OtherClass,C).
```

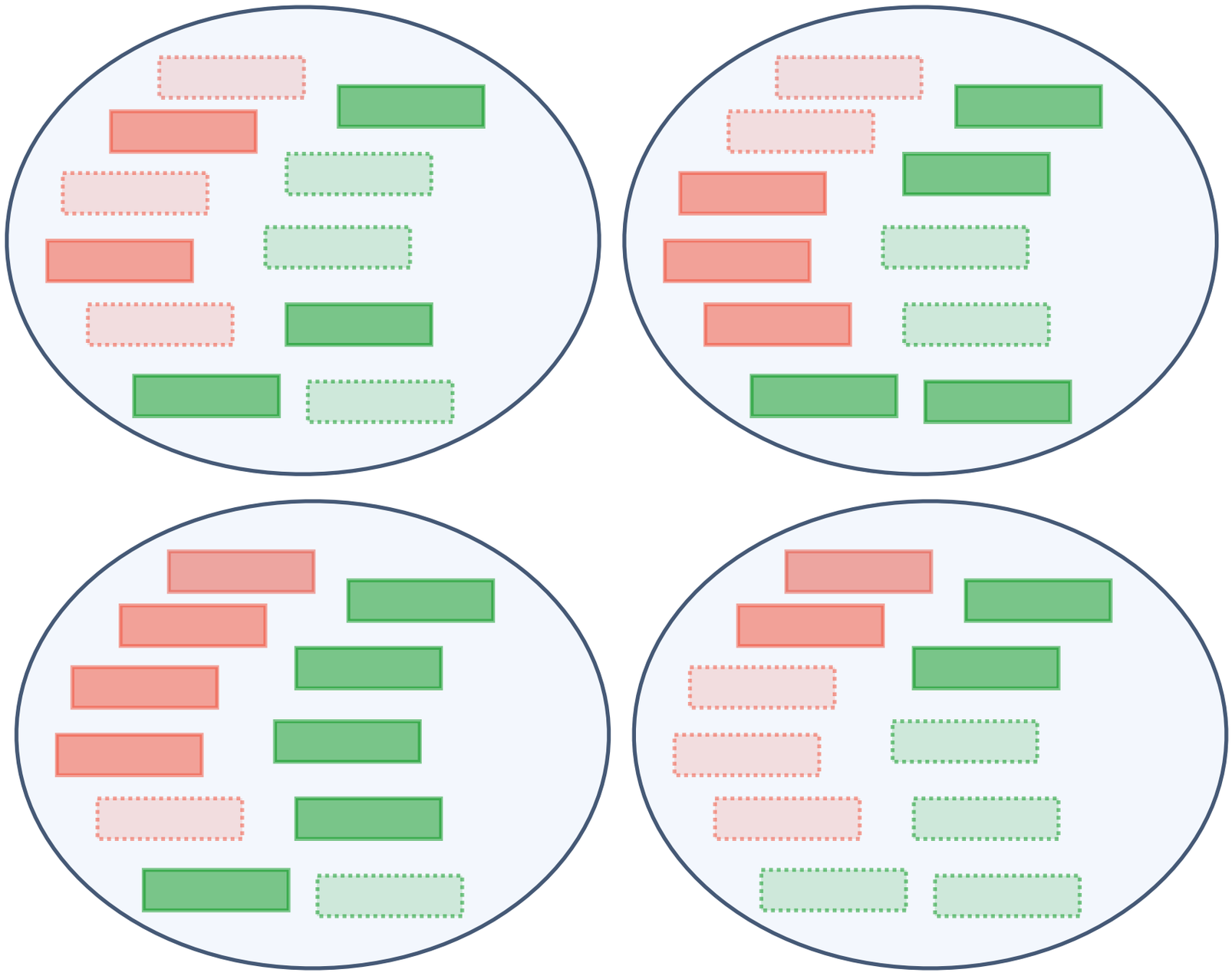
Sampling Interpretations

Sampling Interpretations

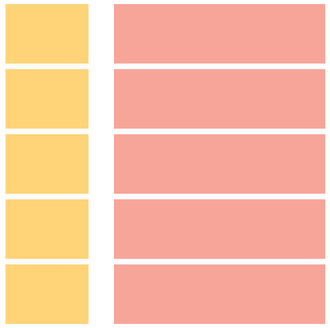


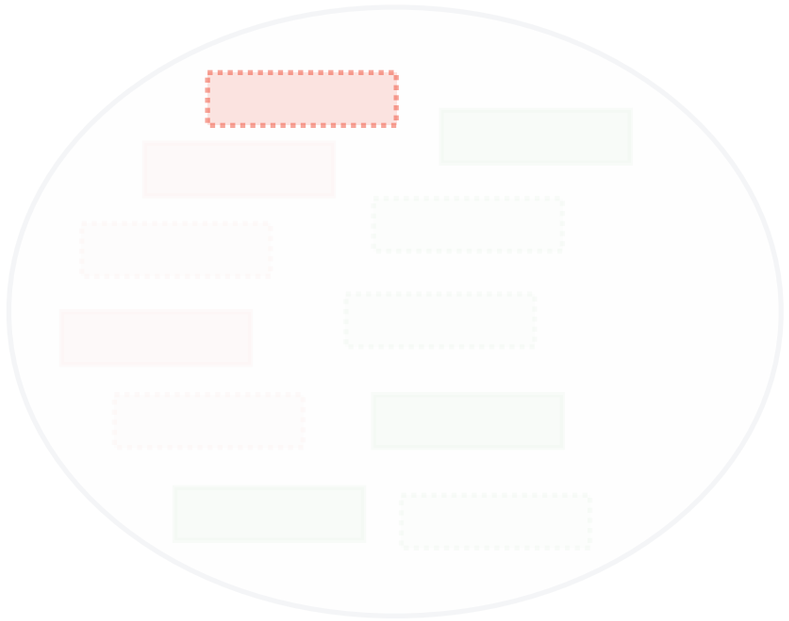


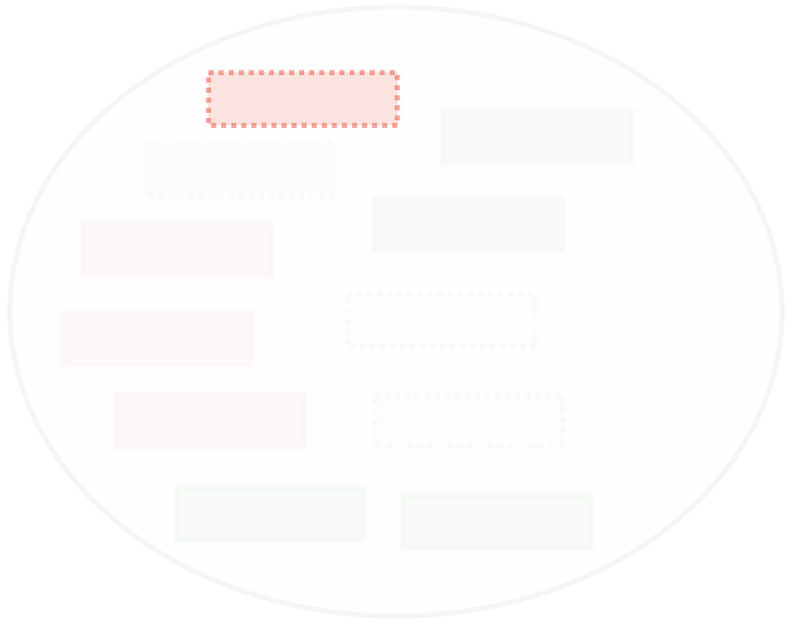
Parameter Estimation

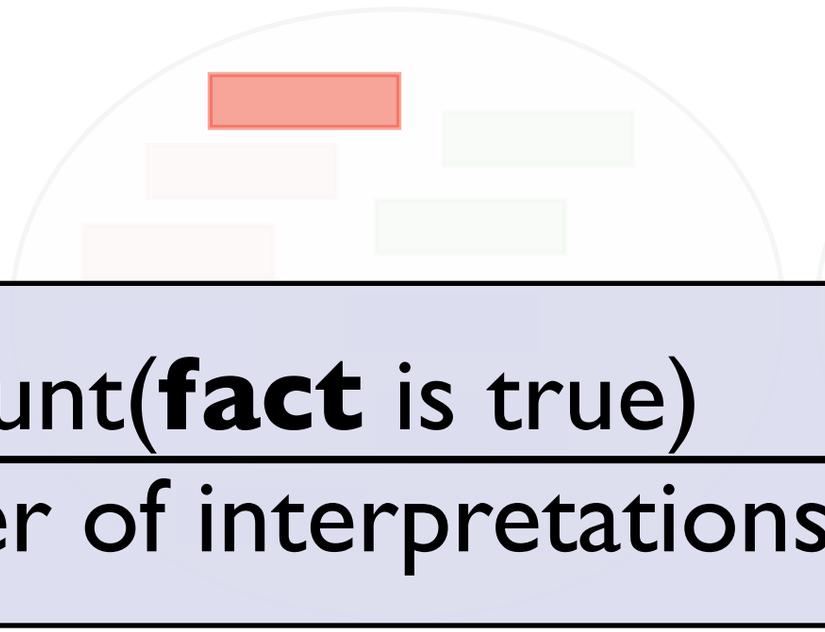


Parameter Estimation



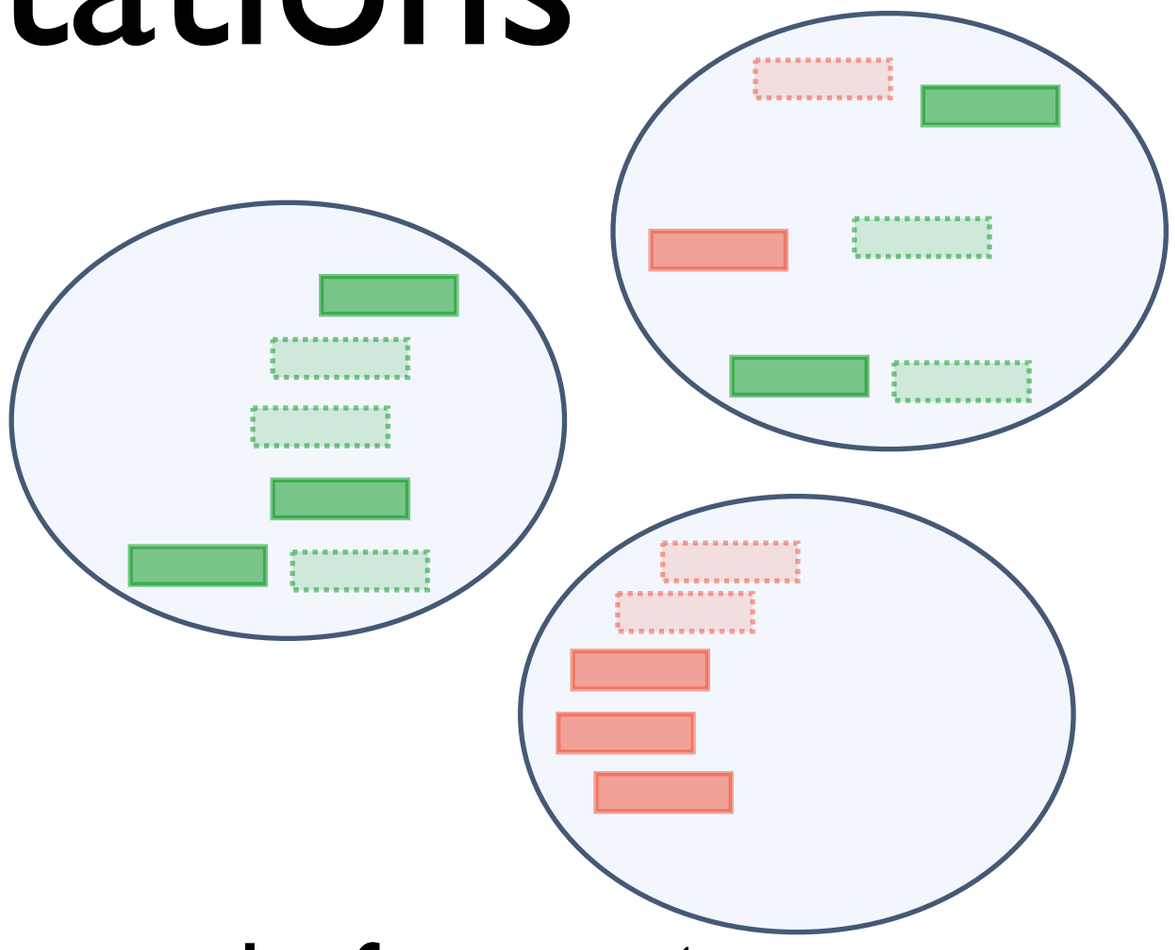






$$p(\textbf{fact}) = \frac{\text{count}(\textbf{fact} \text{ is true})}{\text{Number of interpretations}}$$

Learning from partial interpretations



- Not all facts observed
- Soft-EM
- use **expected count** instead of **count**
- **P(Q |E) -- conditional queries !**

[Gutmann et al, ECML 11; Fierens et al, TPLP 14]

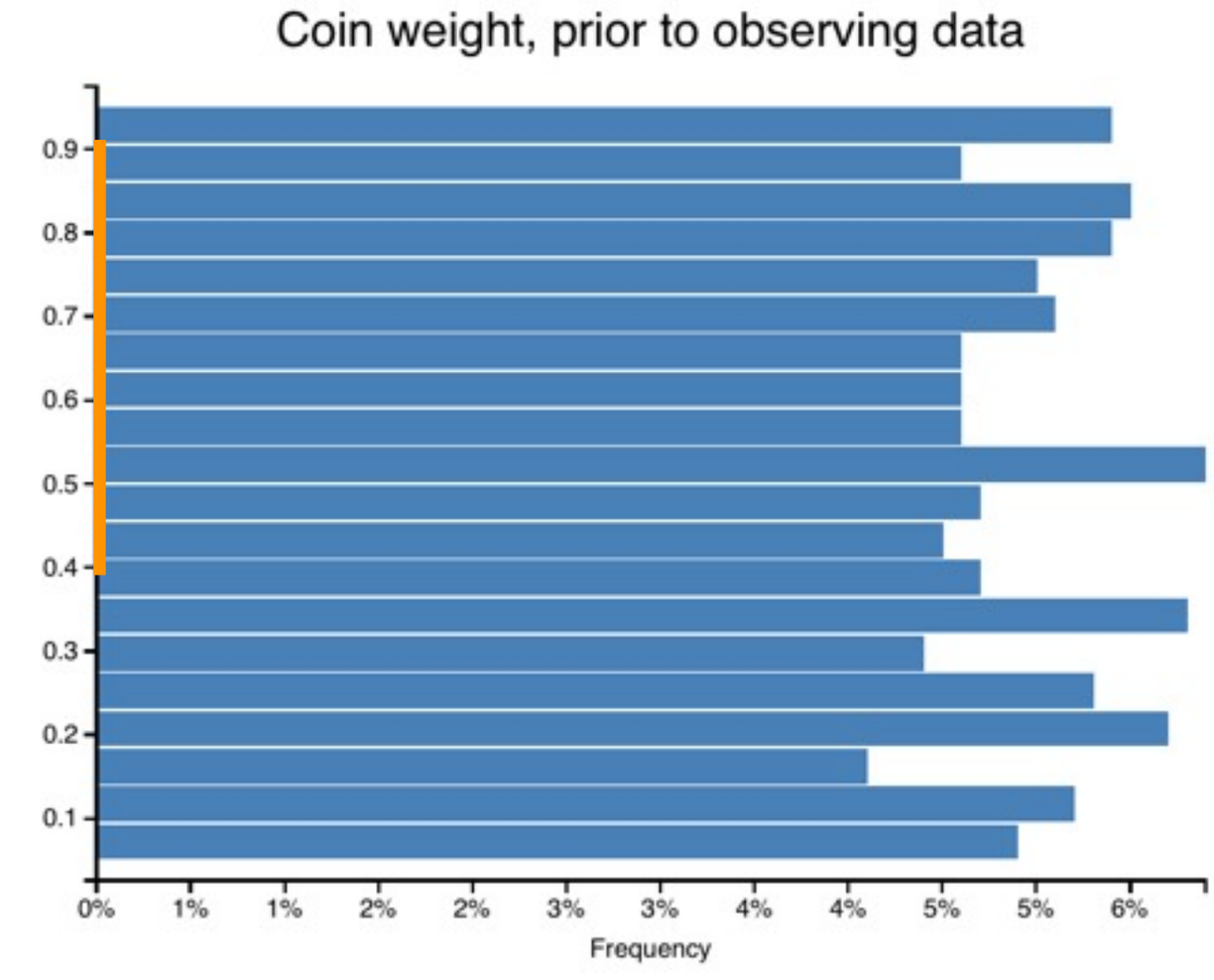
Bayesian Parameter Learning

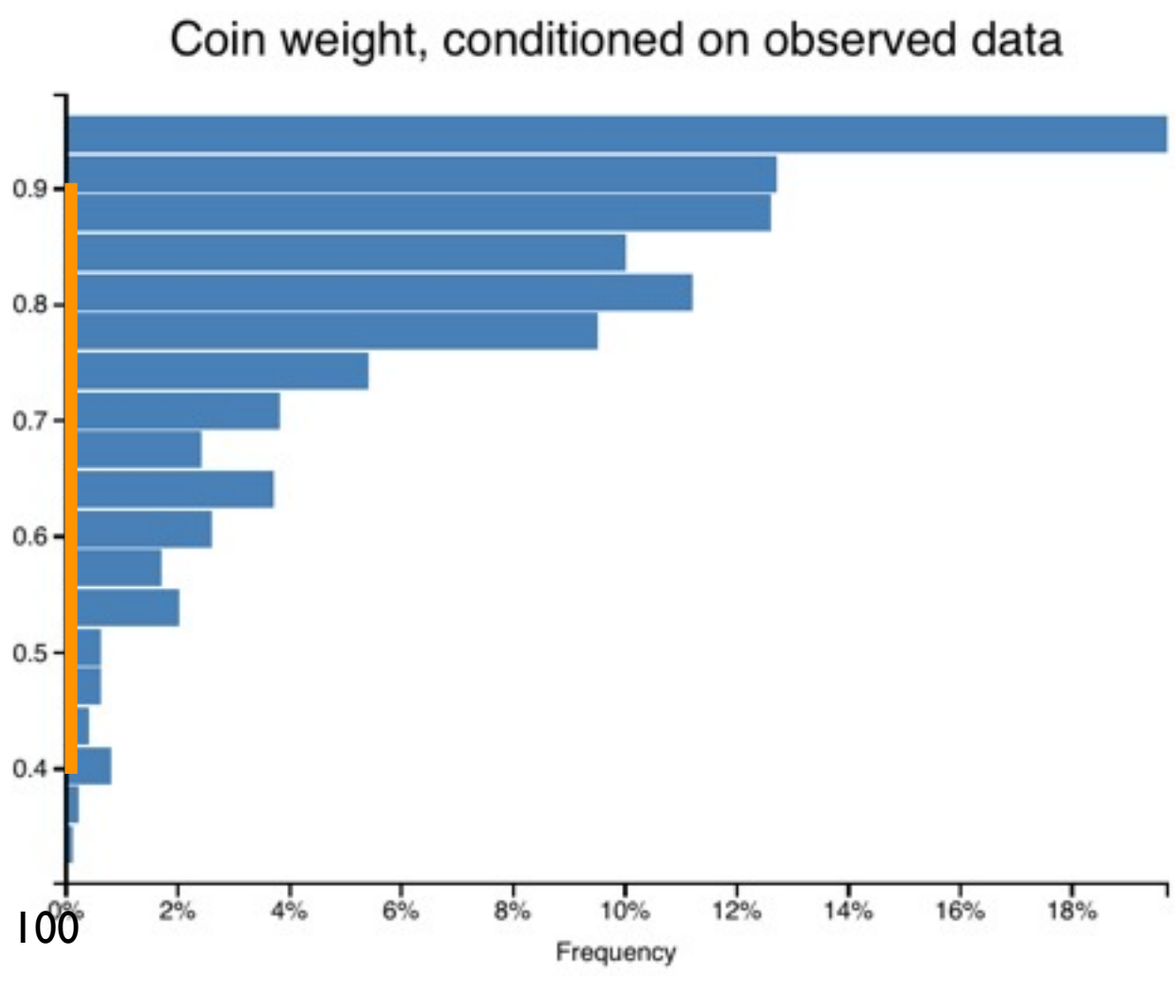
- Learning as inference (e.g., Church)
- Prior distributions for parameters
- Given data, find most likely parameter values

Example

[from probmods.org]

- Flipping a coin with unknown weight
- Prior: uniform distribution on [0,1]
- Observation: 5x heads in a row
- Sampling from Church model:





ProbLog Example

prior

```
0.05::weight(C,0.1); 0.2::weight(C,0.3); 0.5::weight(C,0.5);
                  0.2::weight(C,0.7); 0.05::weight(C,0.9) <- coin(C).
```

```
Param::toss(_,Param,_).
heads(C,R) :- weight(C,Param),toss(C,Param,R).
tails(C,R) :- weight(C,Param),\+toss(C,Param,R).

data(C,[]).
data(C,[h|R]) :- heads(C,R), data(C,R).
data(C,[t|R]) :- tails(C,R), data(C,R).
```

```
coin(c1).
coin(c2).
param(0.1).
param(0.3).
param(0.5).
param(0.7).
param(0.9).
```

```
query(weight(C,X)) :- coin(C),param(X).
```

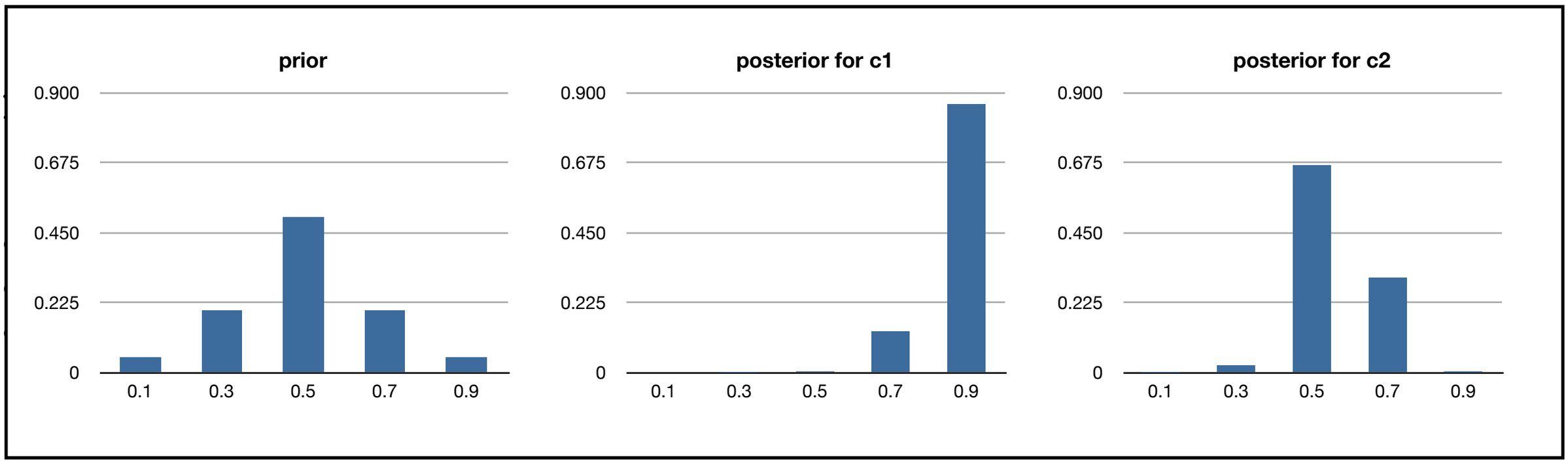
ask for posterior

```
evidence(data(c1,[h,h,h,h,h,h,h,h,h,h,h,h,h,h]),true).
evidence(data(c2,[h,t,h,h,h,h,h,t,t,h,t,t,h]),true).
```

data

ProbLog Example

```
0.05::weight(C,0.1); 0.2::weight(C,0.3); 0.5::weight(C,0.5);
                     0.2::weight(C,0.7); 0.05::weight(C,0.9) <- coin(C).
```

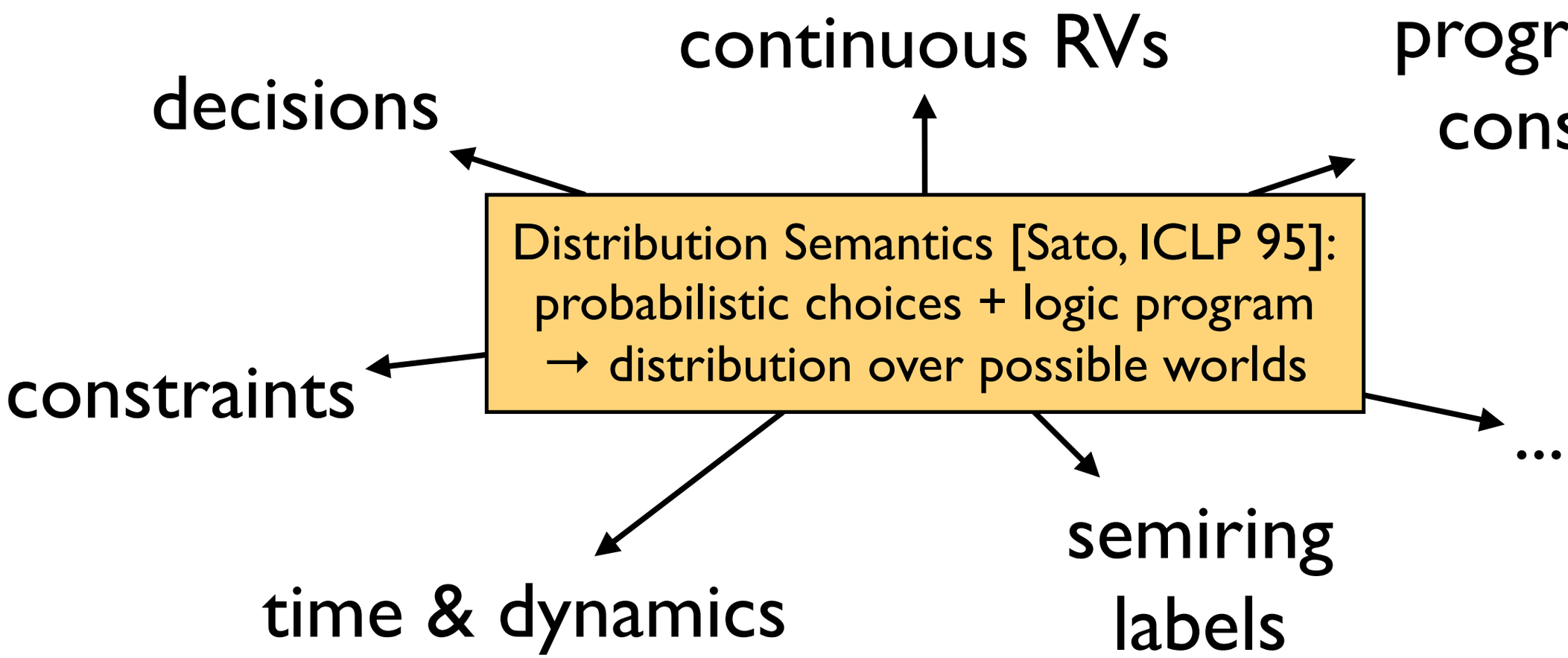



```
query(weight(C,X)) :- coin(C),param(X).

evidence(data(c1,[h,h,h,h,h,h,h,h,h,h,h,h,h,h]),true).
evidence(data(c2,[h,t,h,h,h,h,h,t,t,h,t,t,h]),true).
```

Roadmap

- Modeling (with detours to related work)
- Reasoning (and a bit of learning)
- Language extensions

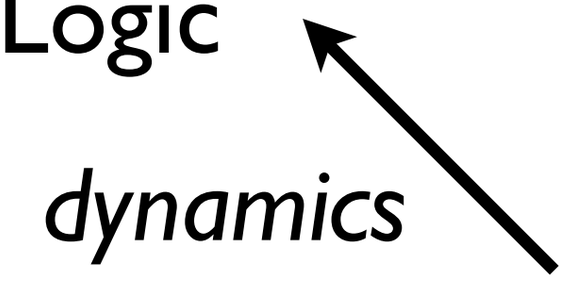
Extensions of basic PLP



Extensions and Variants of ProbLog

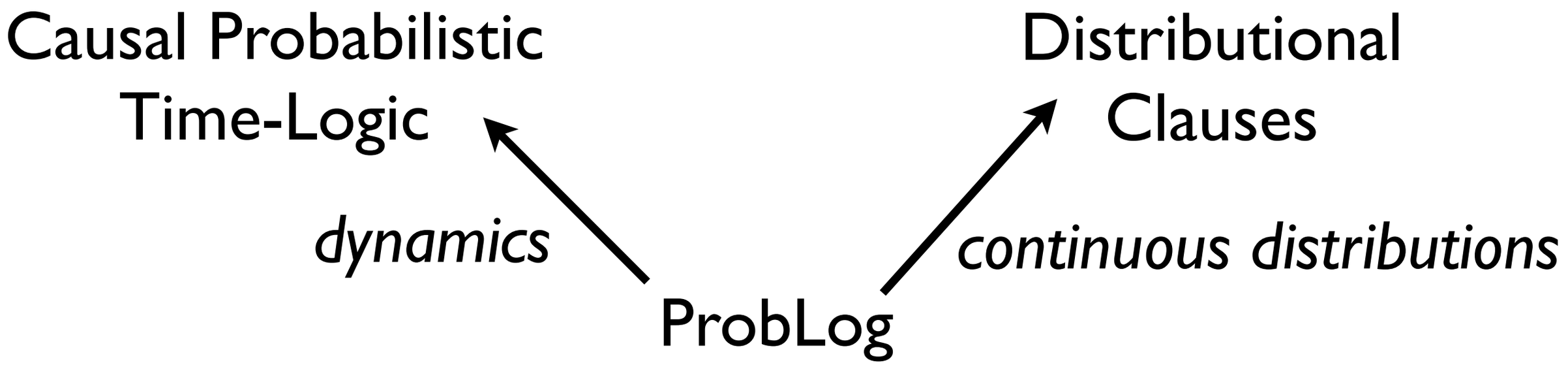
Extensions and Variants of ProbLog

Causal Probabilistic Time-Logic *dynamics*

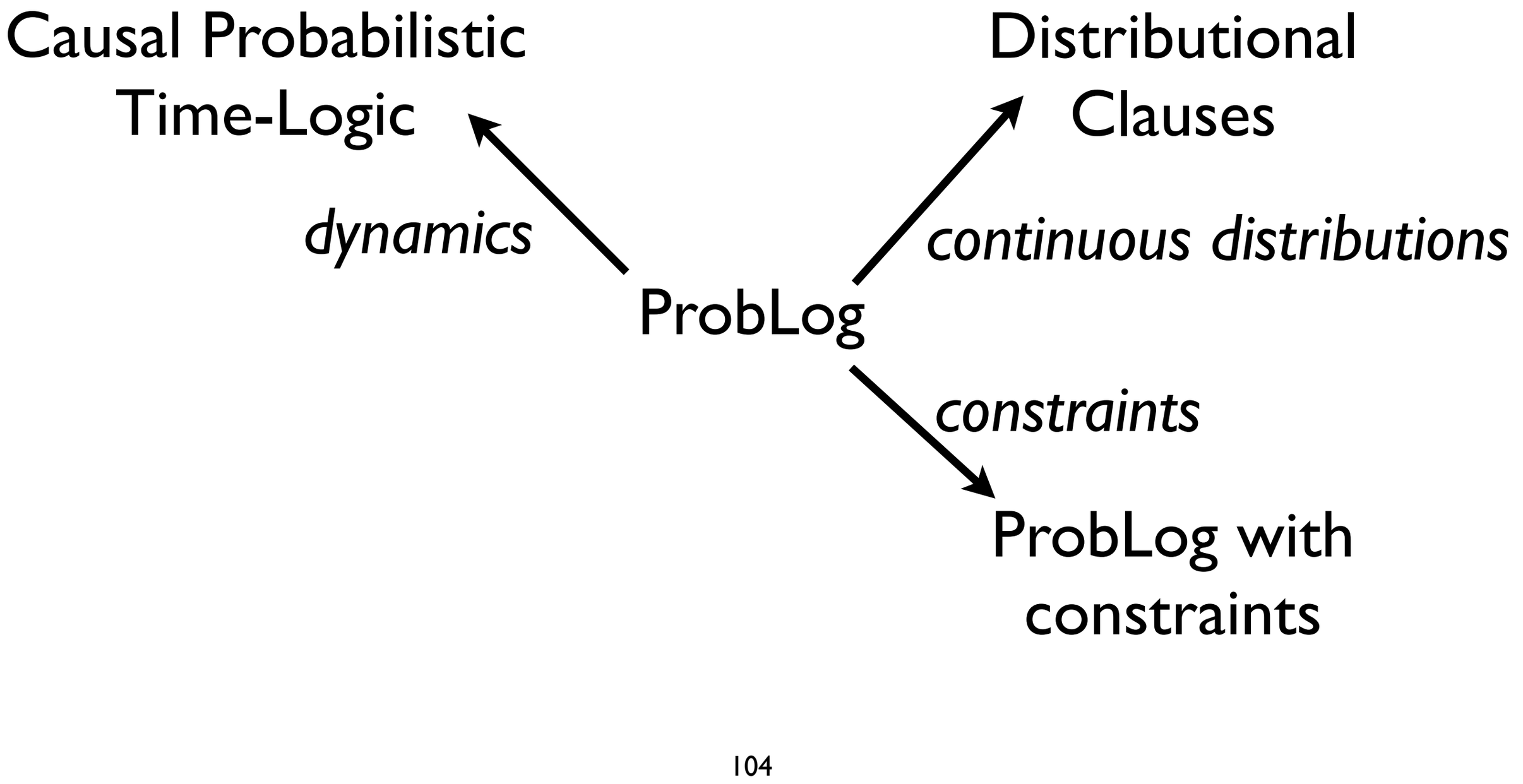


ProbLog

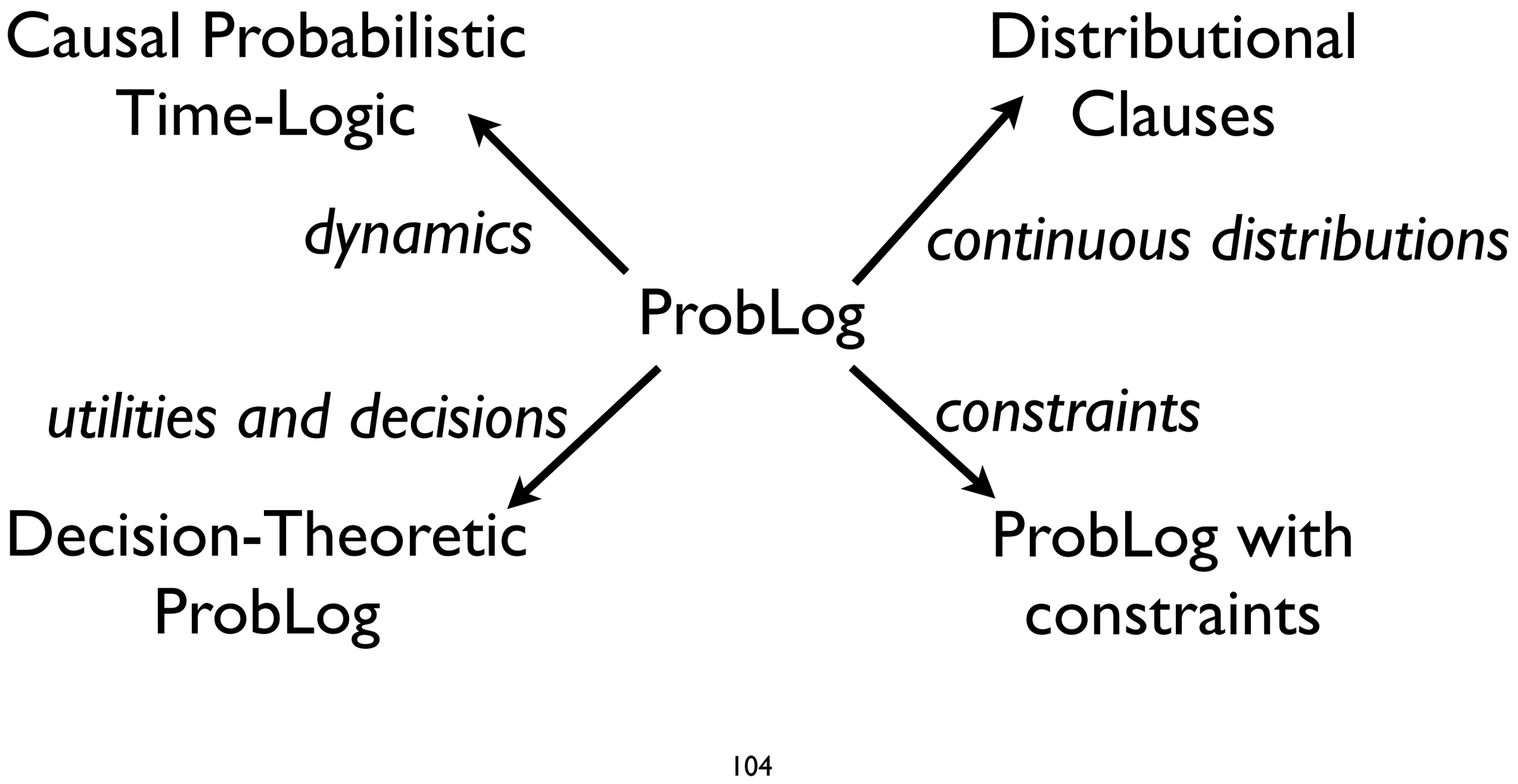
Extensions and Variants of ProbLog



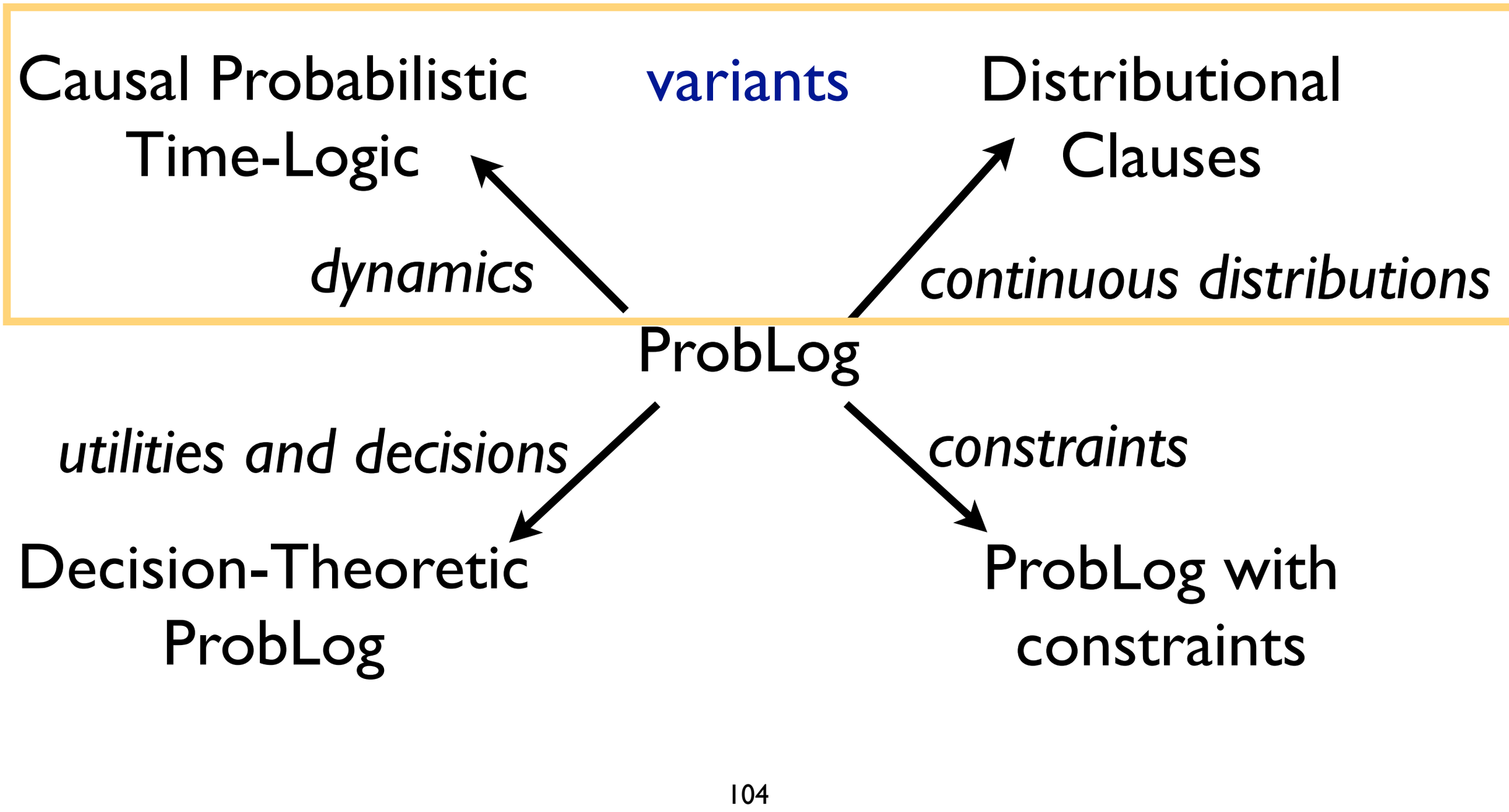
Extensions and Variants of ProbLog



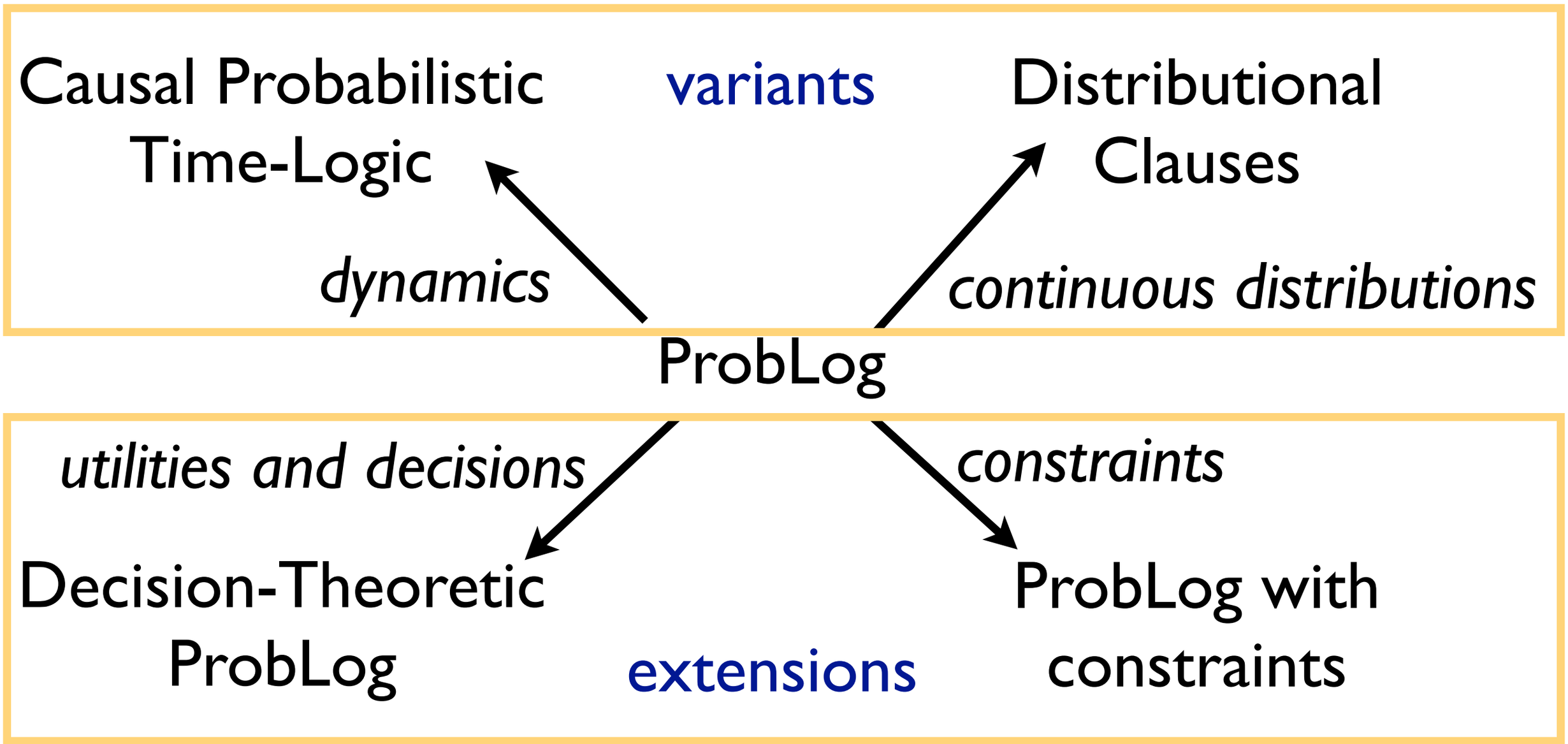
Extensions and Variants of ProbLog



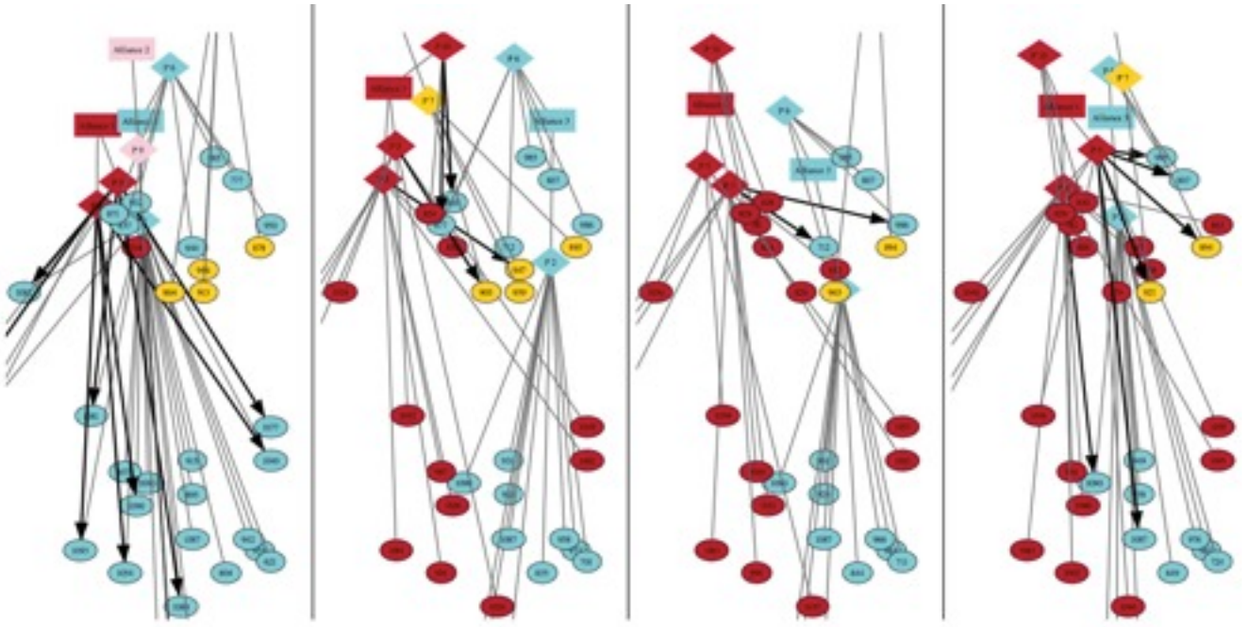
Extensions and Variants of ProbLog



Extensions and Variants of ProbLog



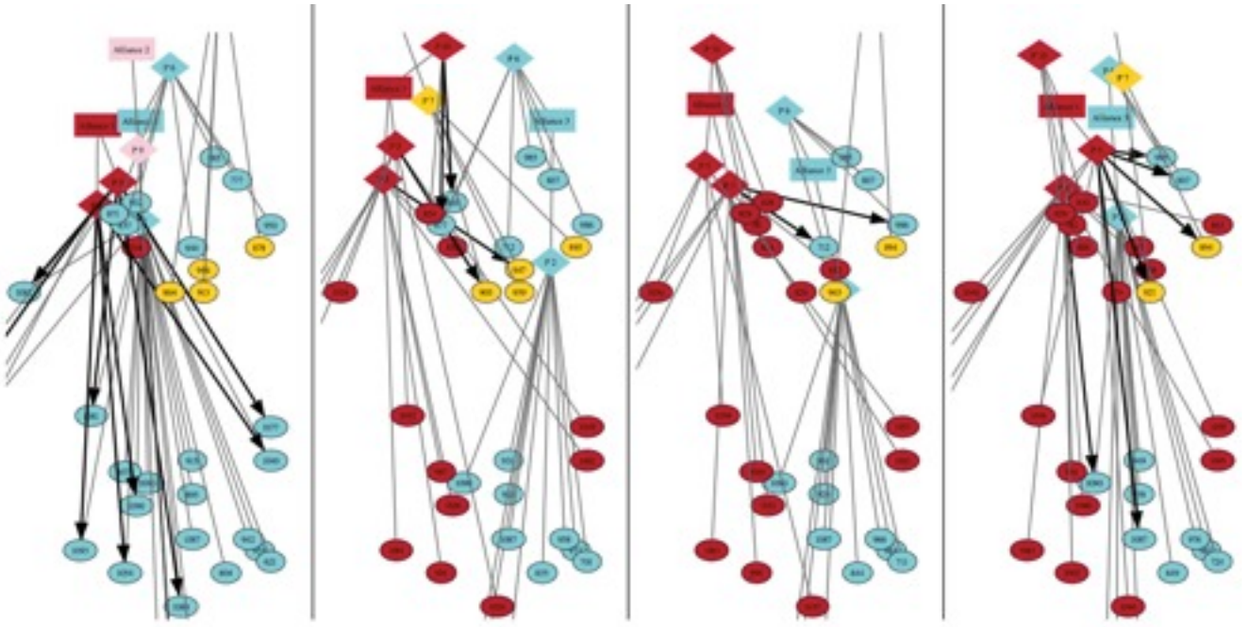
Causal Probabilistic Time-Logic (CPT-L)



how does the world change over time?

[Thon et al, MLJ 11]

Causal Probabilistic Time-Logic (CPT-L)



how does the world change over time?

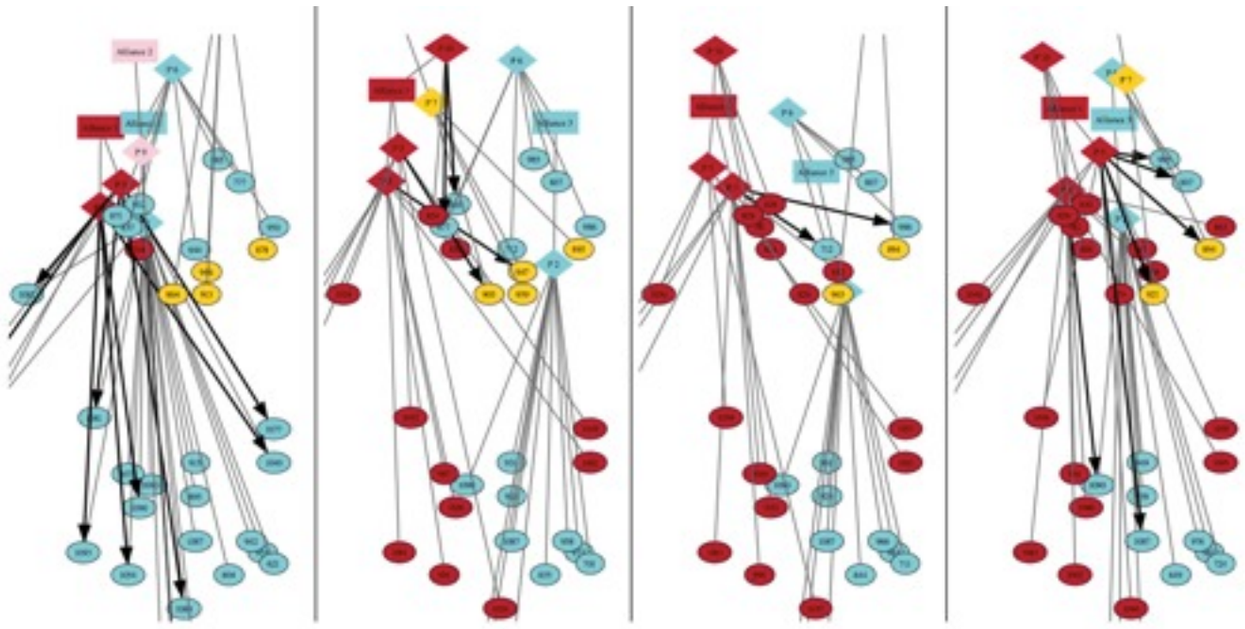
```
0.4::conquest(Attacker,C); 0.6::nil <-
```

```
city(C,Owner),city(C2,Attacker),close(C,C2).
```

if **cause** holds at time T

[Thon et al, MLJ 11]

Causal Probabilistic Time-Logic (CPT-L)



how does the world change over time?

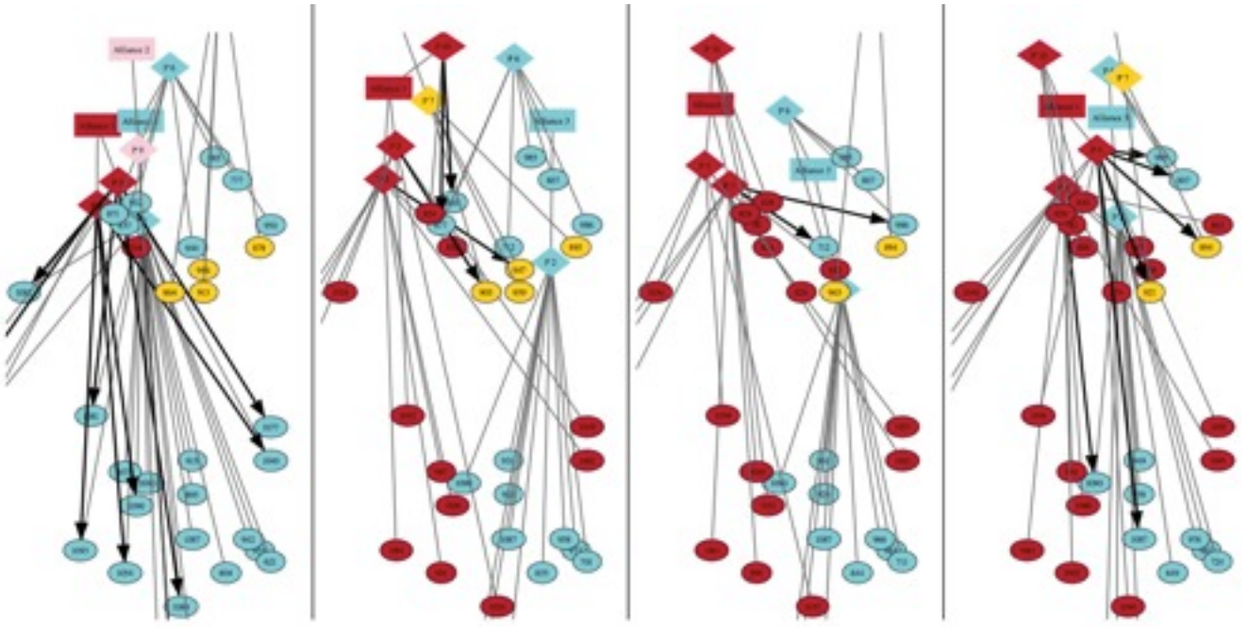
one of the **effects** holds at time T+1

```
0.4::conquest(Attacker,C); 0.6::nil <-
        city(C,Owner),city(C2,Attacker),close(C,C2).
```

if **cause** holds at time T

[Thon et al, MLJ 11]

Causal Probabilistic Time-Logic (CPT-L)



how does the world change over time?

one of the **effects** holds at time T+1

```
0.4::conquest(Attacker,C); 0.6::nil <-

                city(C,Owner),city(C2,Attacker),close(C,C2).
```

if **cause** holds at time T

[Thon et al, MLJ 11]

Distributional Clauses (DC)

- Discrete- and continuous-valued random variables
- Inference: particle filter

[Gutmann et al, TPLP 11; Nitti et al, IROS 13]

Distributional Clauses (DC)

- Discrete- and continuous-valued random variables
- Inference: particle filter

random variable with Gaussian distribution

```
length(Obj) ~ gaussian(6.0,0.45) :- type(Obj,glass).
```

[Gutmann et al, TPLP 11; Nitti et al, IROS 13]

Distributional Clauses (DC)

- Discrete- and continuous-valued random variables
- Inference: particle filter

```
length(Obj) ~ gaussian(6.0,0.45) :- type(Obj,glass).
stackable(OBot,OTop) :-
    ≃length(OBot) ≥ ≃length(OTop),
    ≃width(OBot) ≥ ≃width(OTop).
```

comparing values of random variables

[Gutmann et al, TPLP 11; Nitti et al, IROS 13]

Distributional Clauses (DC)

- Discrete- and continuous-valued random variables
- Inference: particle filter

```
length(Obj) ~ gaussian(6.0,0.45) :- type(Obj,glass).
stackable(OBot,OTop) :-
      ≃length(OBot) ≥ ≃length(OTop),
      ≃width(OBot) ≥ ≃width(OTop).
ontype(Obj,plate) ~ finite([0 : glass, 0.0024 : cup,
                            0 : pitcher, 0.8676 : plate,
                            0.0284 : bowl, 0 : serving,
                            0.1016 : none])
                          :- obj(Obj), on(Obj,O2), type(O2,plate).
```

random variable with discrete distribution

[Gutmann et al, TPLP 11; Nitti et al, IROS 13]

Distributional Clauses (DC)

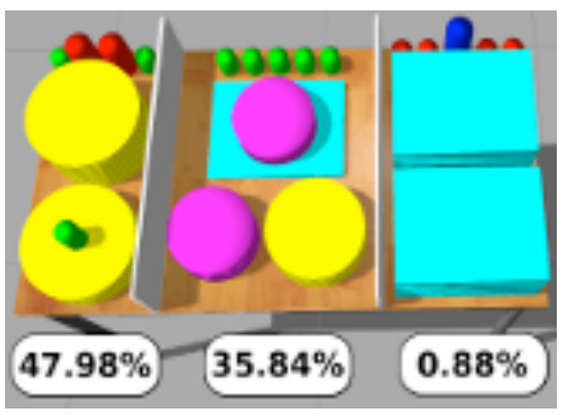
- Discrete- and continuous-valued random variables
- Inference: particle filter

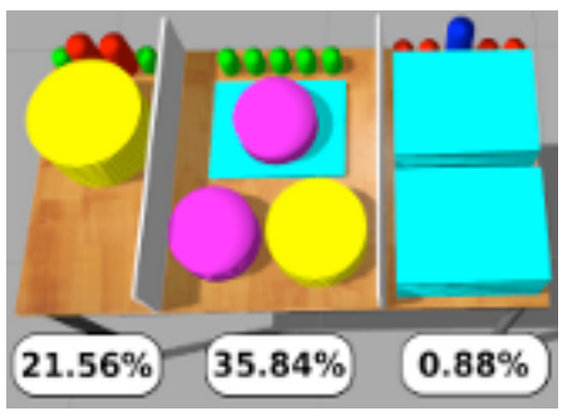
```
length(Obj) ~ gaussian(6.0,0.45) :- type(Obj,glass).
stackable(OBot,OTop) :-
      ≃length(OBot) ≥ ≃length(OTop),
      ≃width(OBot) ≥ ≃width(OTop).
ontype(Obj,plate) ~ finite([0 : glass, 0.0024 : cup,
                            0 : pitcher, 0.8676 : plate,
                            0.0284 : bowl, 0 : serving,
                            0.1016 : none])
                    :- obj(Obj), on(Obj,O2), type(O2,plate).
```

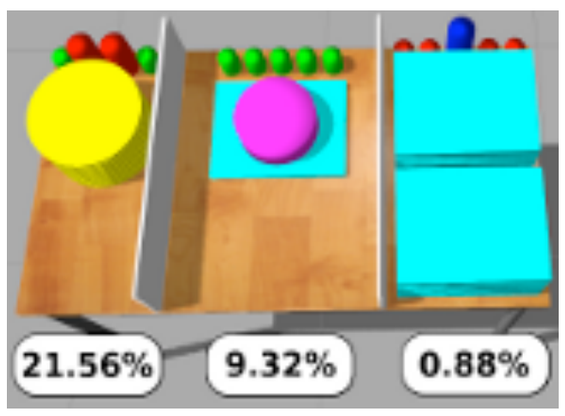
[Gutmann et al, TPLP 11; Nitti et al, IROS 13]

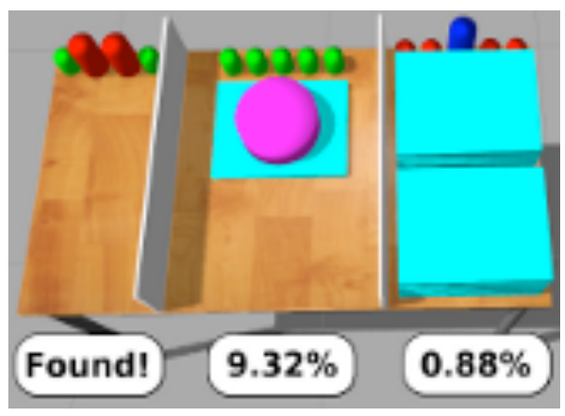
Occluded Object Search

- DC model of objects and their spatial arrangement
- different types of objects suitable for different tasks
- shelves with objects of different shape and size
- given a task, find an object to perform that task







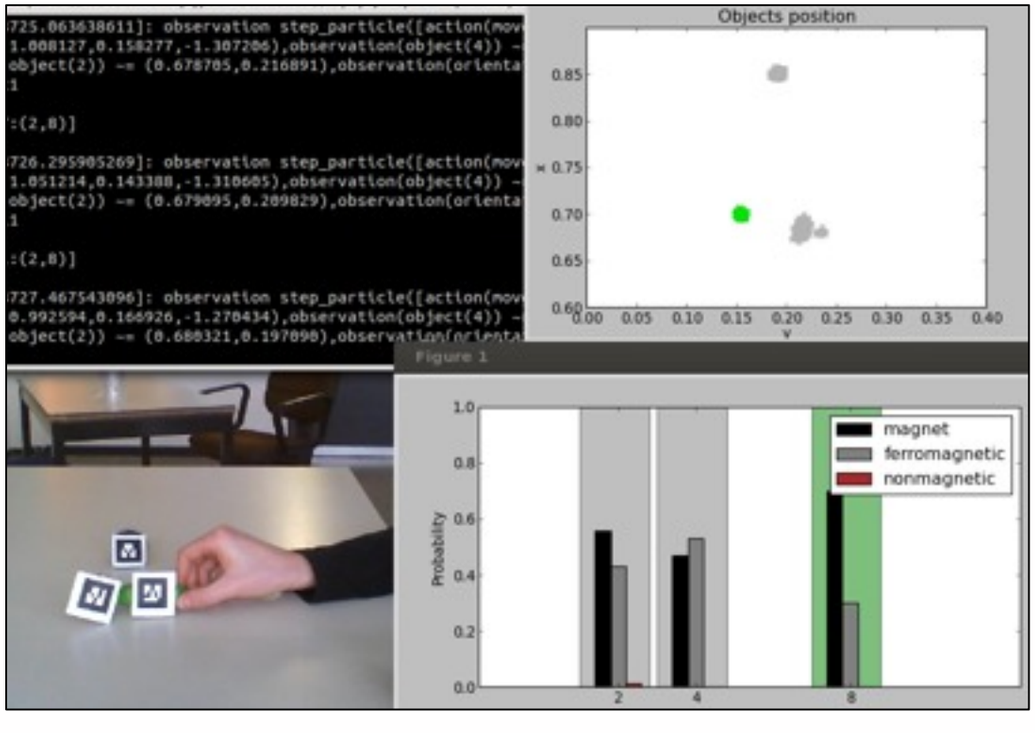


[Moldovan et al, ICRA 12, 14]

Relational State Estimation over Time

Magnetism scenario

- object tracking
- category estimation from interactions

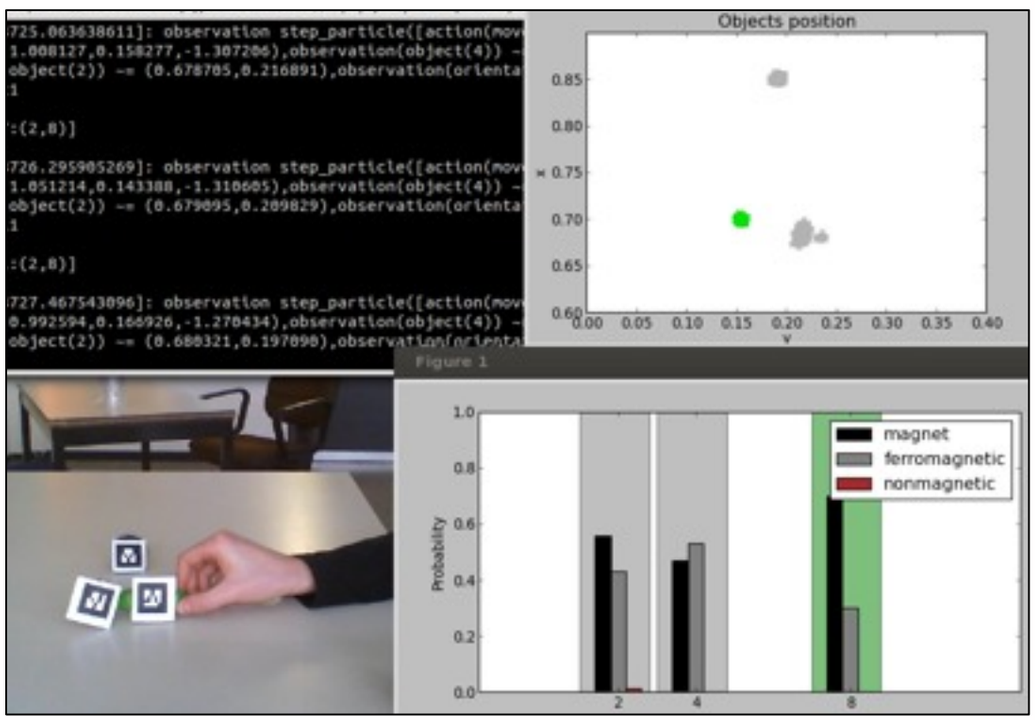


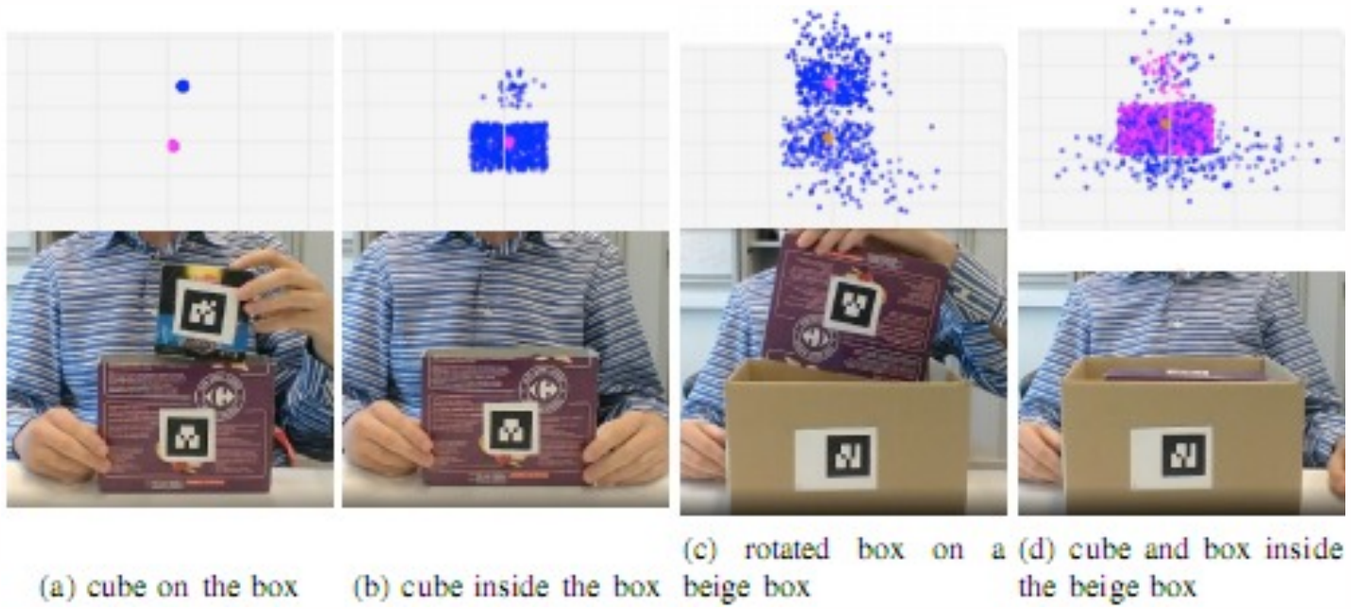
[Nitti et al, IROS 13]

Relational State Estimation over Time

Magnetism scenario

- object tracking
- category estimation from interactions



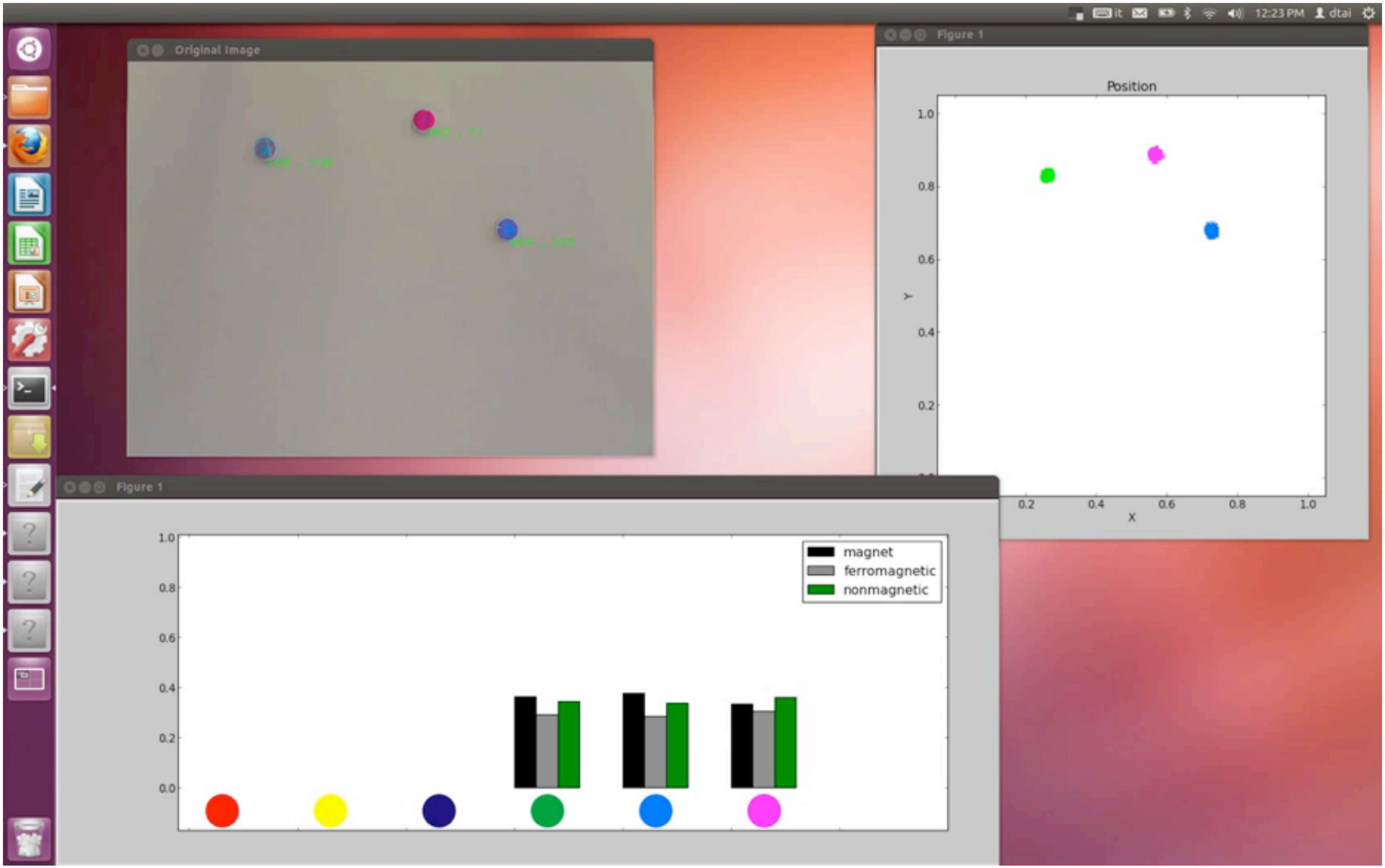


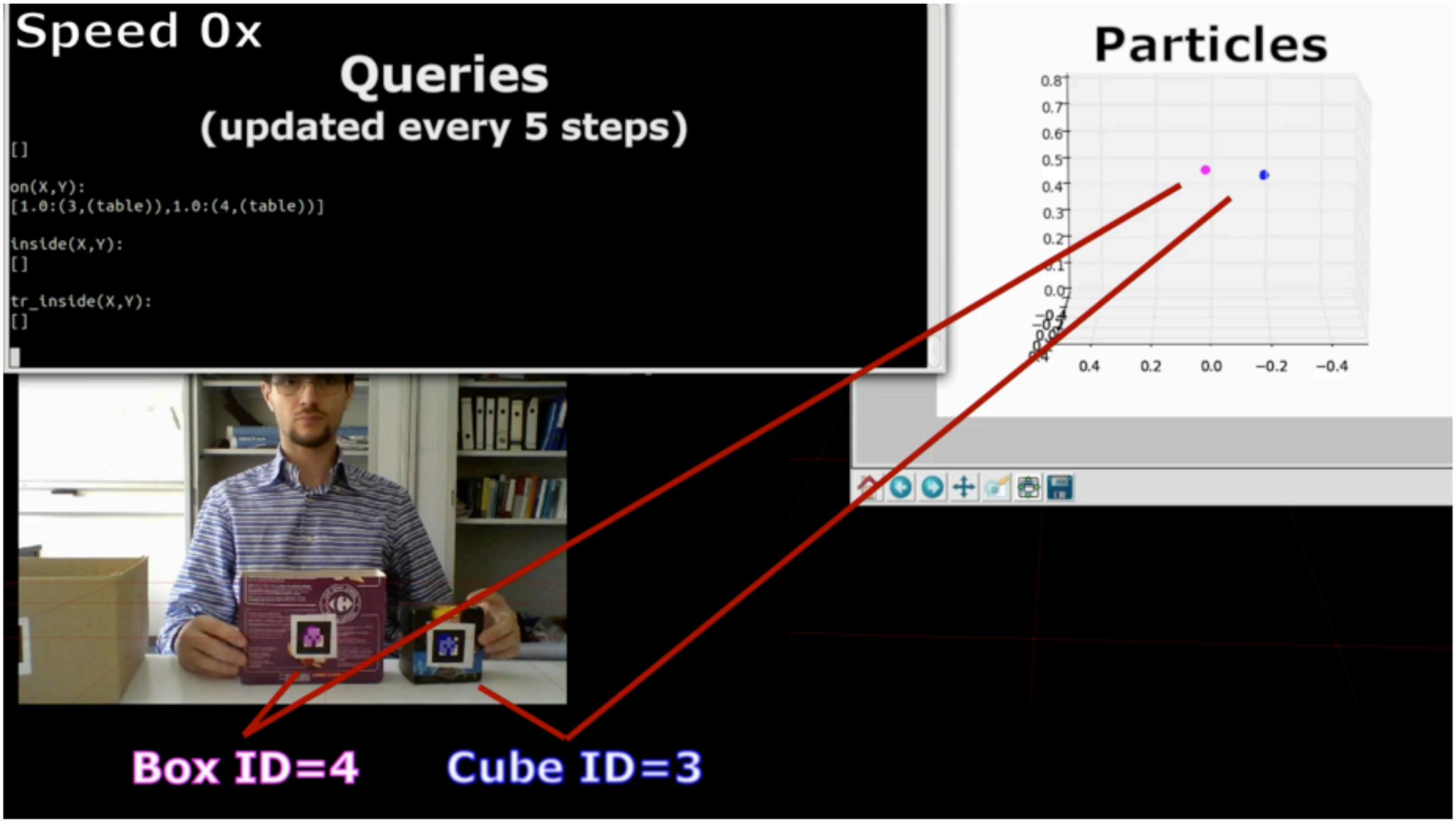
(a) cube on the box (b) cube inside the box (c) rotated box on a beige box (d) cube and box inside the beige box

Box scenario

- object tracking even when invisible
- estimate spatial relations

[Nitti et al, IROS 13]


Original Image
Figure 1
Position
1.0
0.8
0.6
0.4
0.2
Y
0.2
0.4
0.6
0.8
1.0
X
Figure 1
1.0
0.8
0.6
0.4
0.2
0.0
magnet
ferromagnetic
nonmagnetic


Speed 0x
Queries
(updated every 5 steps)
[]
on(X,Y):
[1.0:(3,(table)),1.0:(4,(table))]
inside(X,Y):
[]
tr_inside(X,Y):
[]
Particles
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.4
0.2
0.0
−0.2
−0.4
Box ID=4
Cube ID=3

IROS 13

cProbLog: constraints on possible worlds

```
weight(skis,6).
weight(boots,4).
weight(helmet,3).
weight(gloves,2).

P::pack(Item) :-
  weight(Item,Weight),
  P is 1.0/Weight.

excess(Limit) :- ...

not excess(10).
pack(helmet) v pack(boots).
```


[Fierens et al, PP 12; Shterionov et al]

cProbLog: constraints on possible worlds

```
weight(skis,6).
weight(boots,4).
weight(helmet,3).
weight(gloves,2).
```

distribution over **all** possible worlds

```
P::pack(Item) :-
  weight(Item,Weight),
  P is 1.0/Weight.

excess(Limit) :- ...

not excess(10).
pack(helmet) v pack(boots).
```

sbhg e(10)	sb g e(10)	sbh e(10)	sb
s hg e(10)	s g	s h	s
bhg	b g	bh	b
hg	g	h	

[Fierens et al, PP 12; Shterionov et al]

cProbLog: constraints on possible worlds

```
weight(skis,6).
weight(boots,4).
weight(helmet,3).
weight(gloves,2).

P::pack(Item) :-
  weight(Item,Weight),
  P is 1.0/Weight.

excess(Limit) :- ...

not excess(10).
pack(helmet) v pack(boots).
```

constraints as first-order logic formulas

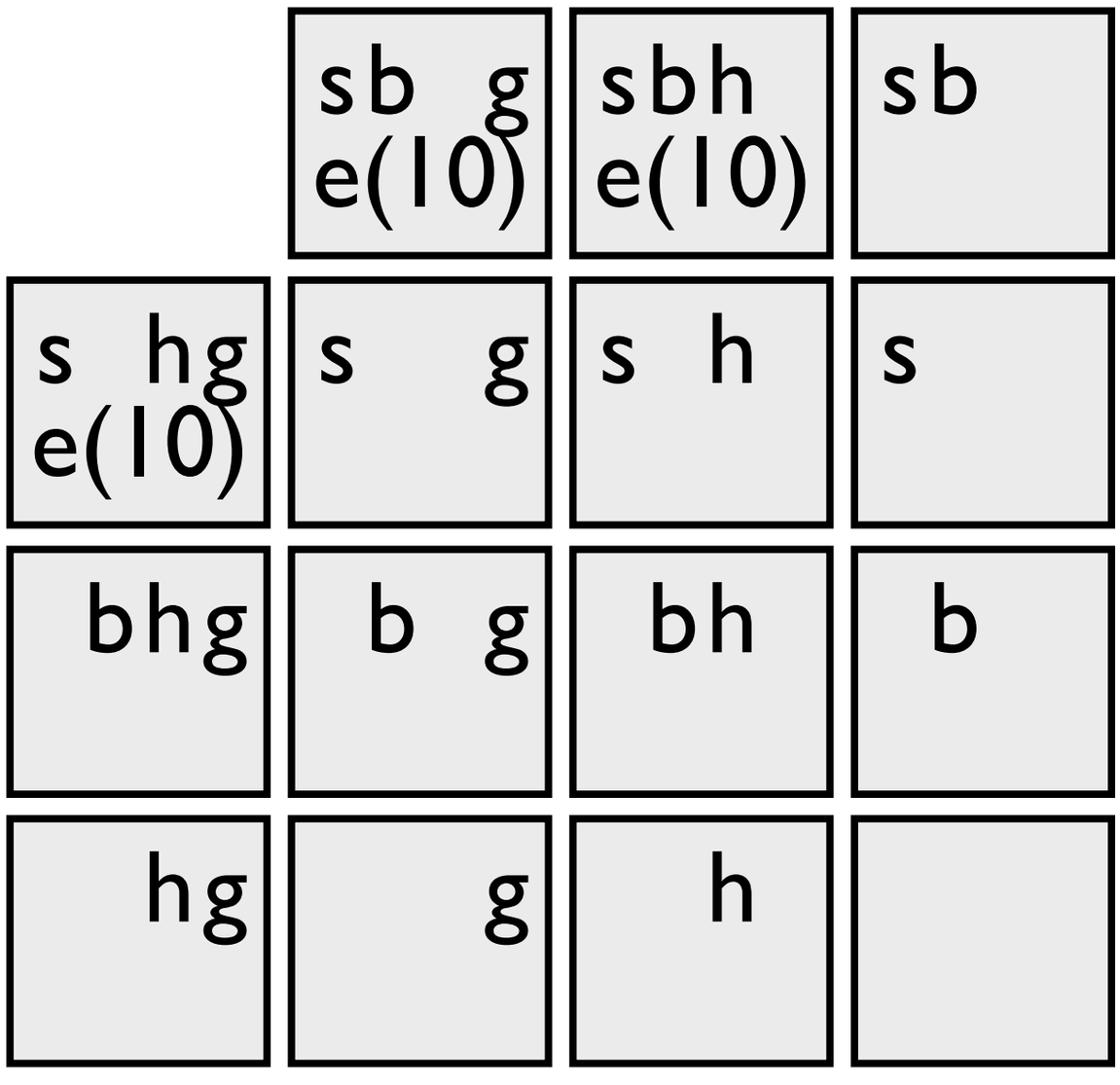
sbhg e(10)	sb g e(10)	sbh e(10)	sb
s hg e(10)	s g	s h	s
bhg	b g	bh	b
hg	g	h	

[Fierens et al, PP 12; Shterionov et al]

cProbLog: constraints on possible worlds

```
weight(skis,6).
weight(boots,4).
weight(helmet,3).
weight(gloves,2).

P::pack(Item) :-
  weight(Item,Weight),
  P is 1.0/Weight.

excess(Limit) :- ...

not excess(10).
pack(helmet) v pack(boots).
```

constraints as first-order logic formulas

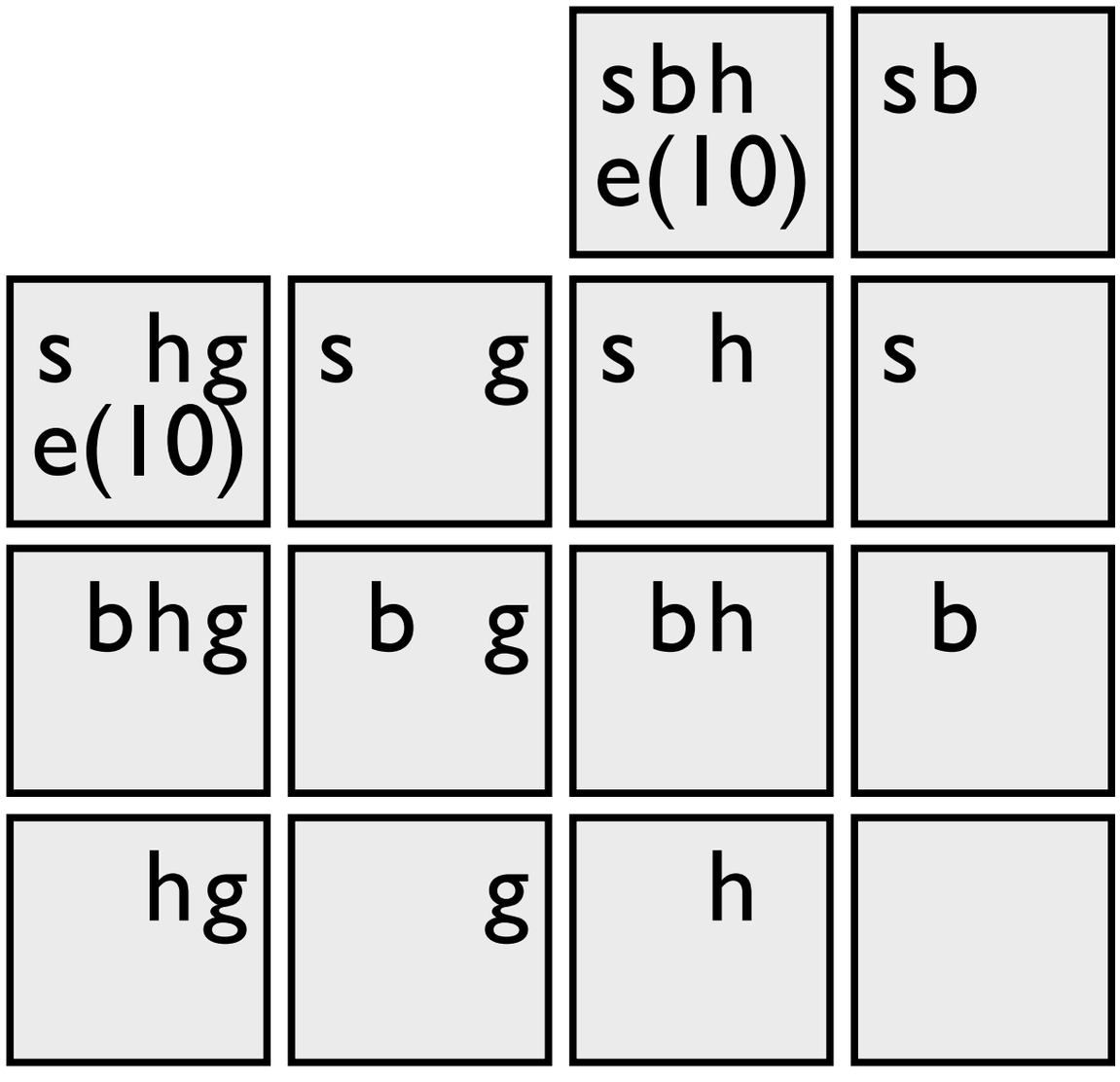


[Fierens et al, PP 12; Shterionov et al]

cProbLog: constraints on possible worlds

```
weight(skis,6).
weight(boots,4).
weight(helmet,3).
weight(gloves,2).

P::pack(Item) :-
  weight(Item,Weight),
  P is 1.0/Weight.

excess(Limit) :- ...

not excess(10).
pack(helmet) v pack(boots).
```

constraints as first-order logic formulas

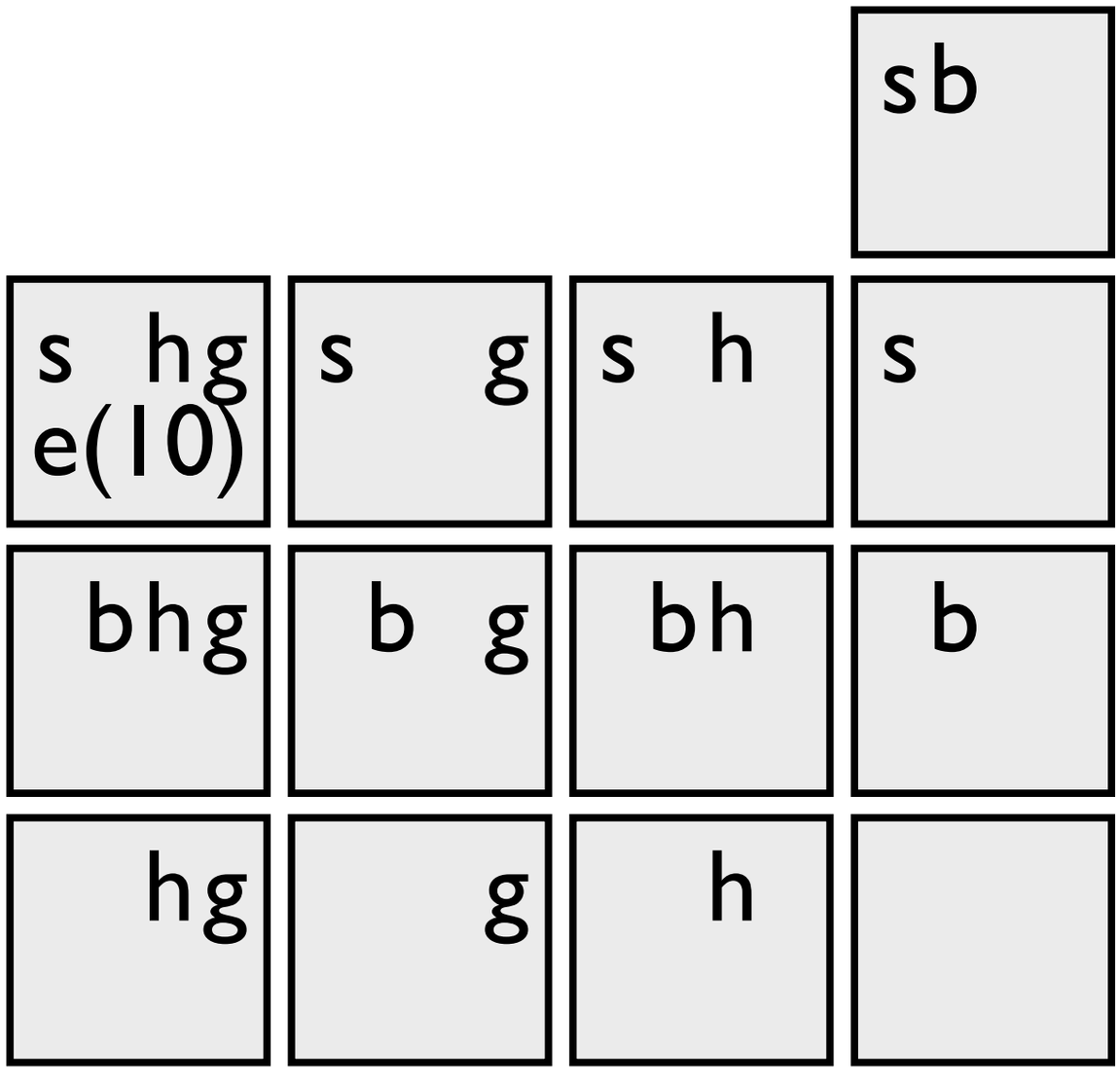


[Fierens et al, PP 12; Shterionov et al]

cProbLog: constraints on possible worlds

```
weight(skis,6).
weight(boots,4).
weight(helmet,3).
weight(gloves,2).

P::pack(Item) :-
  weight(Item,Weight),
  P is 1.0/Weight.

excess(Limit) :- ...

not excess(10).
pack(helmet) v pack(boots).
```

constraints as first-order logic formulas

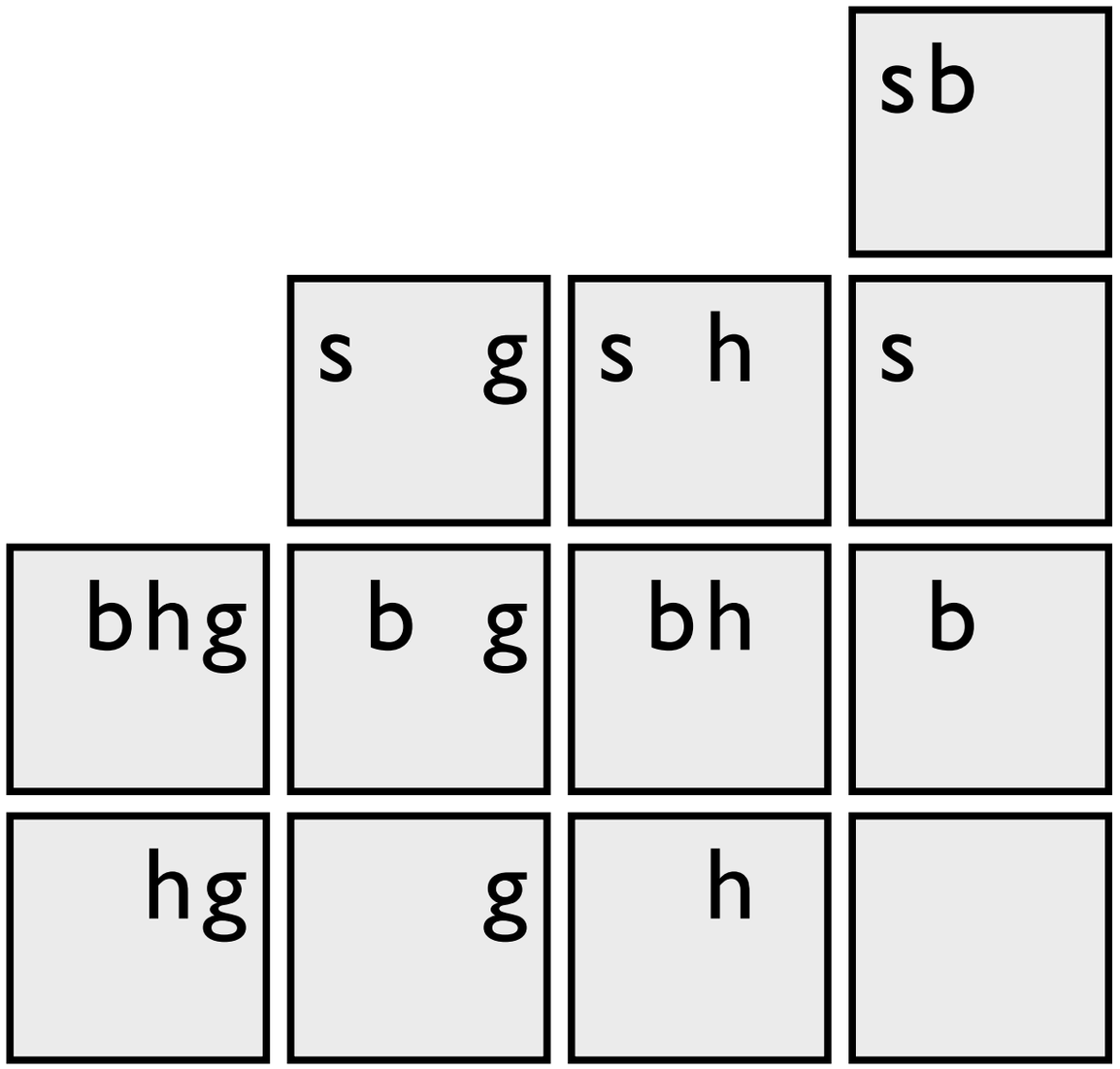


[Fierens et al, PP 12; Shterionov et al]

cProbLog: constraints on possible worlds

```
weight(skis,6).
weight(boots,4).
weight(helmet,3).
weight(gloves,2).

P::pack(Item) :-
  weight(Item,Weight),
  P is 1.0/Weight.

excess(Limit) :- ...

not excess(10).
pack(helmet) v pack(boots).
```

constraints as first-order logic formulas



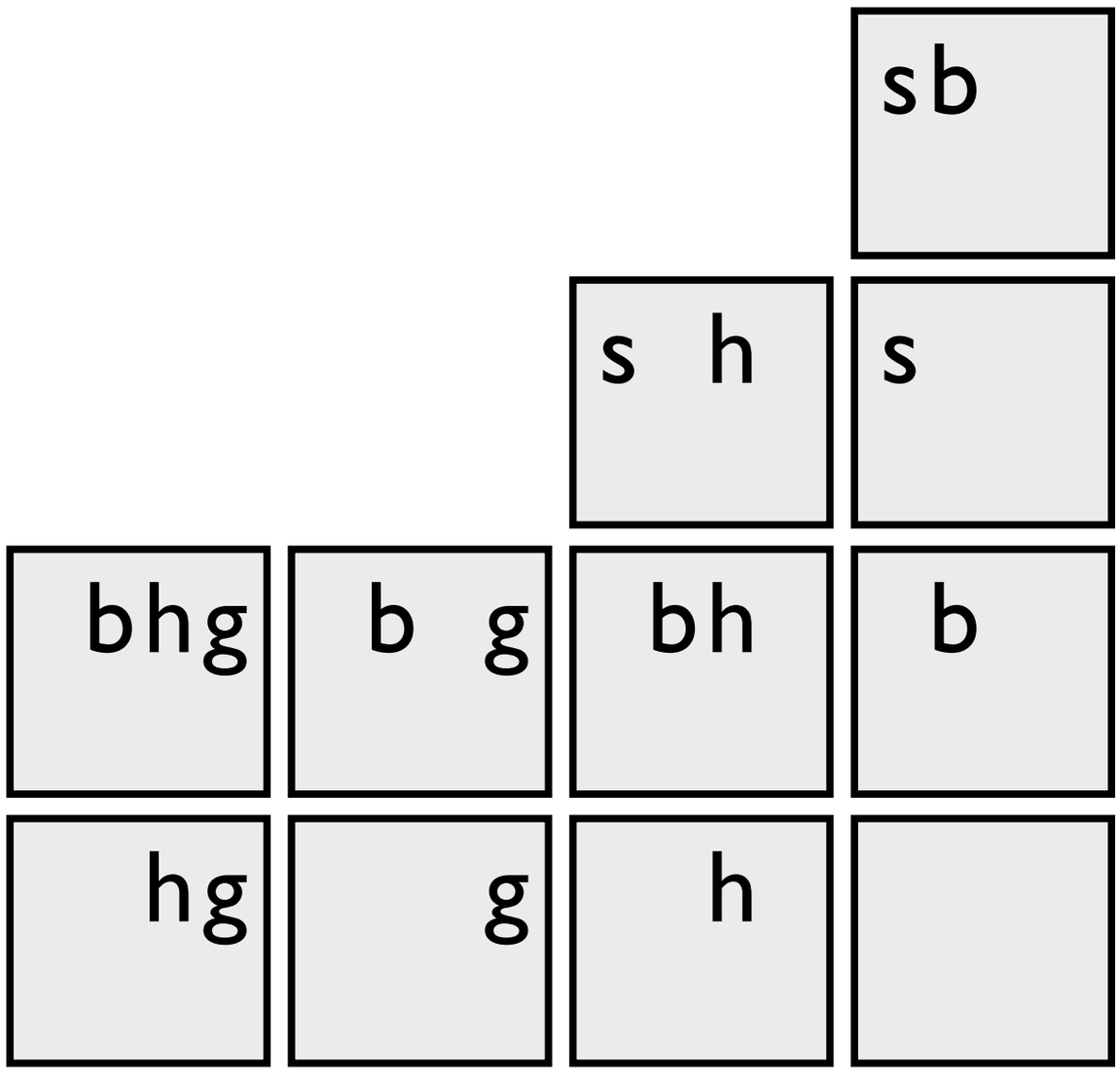
[Fierens et al, PP 12; Shterionov et al]

cProbLog: constraints on possible worlds

```
weight(skis,6).
weight(boots,4).
weight(helmet,3).
weight(gloves,2).

P::pack(Item) :-
  weight(Item,Weight),
  P is 1.0/Weight.

excess(Limit) :- ...
```

```
not excess(10).
pack(helmet) v pack(boots).
```

constraints as first-order logic formulas



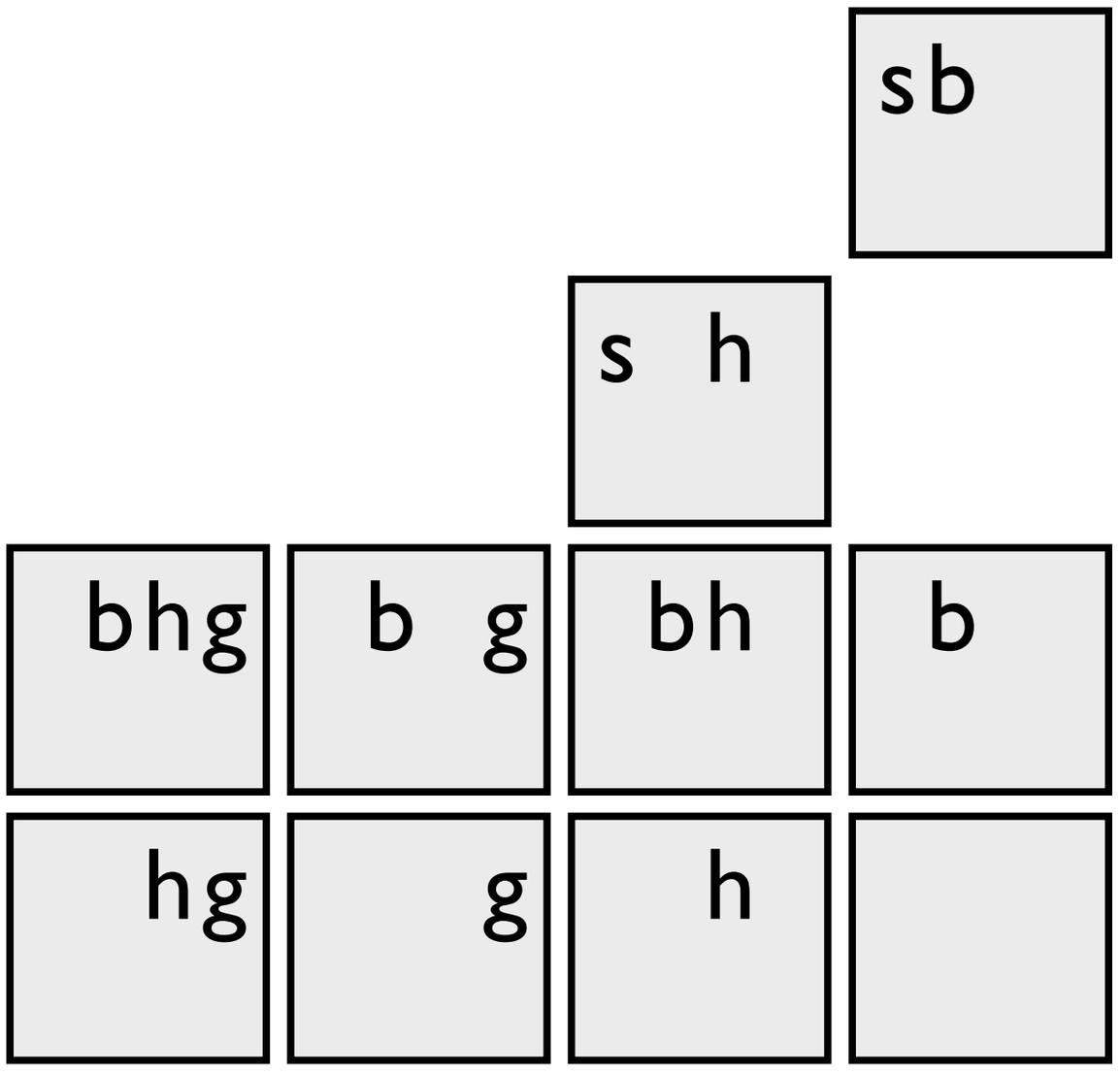
[Fierens et al, PP 12; Shterionov et al]

cProbLog: constraints on possible worlds

```
weight(skis,6).
weight(boots,4).
weight(helmet,3).
weight(gloves,2).

P::pack(Item) :-
  weight(Item,Weight),
  P is 1.0/Weight.

excess(Limit) :- ...
```

```
not excess(10).
pack(helmet) v pack(boots).
```

constraints as first-order logic formulas



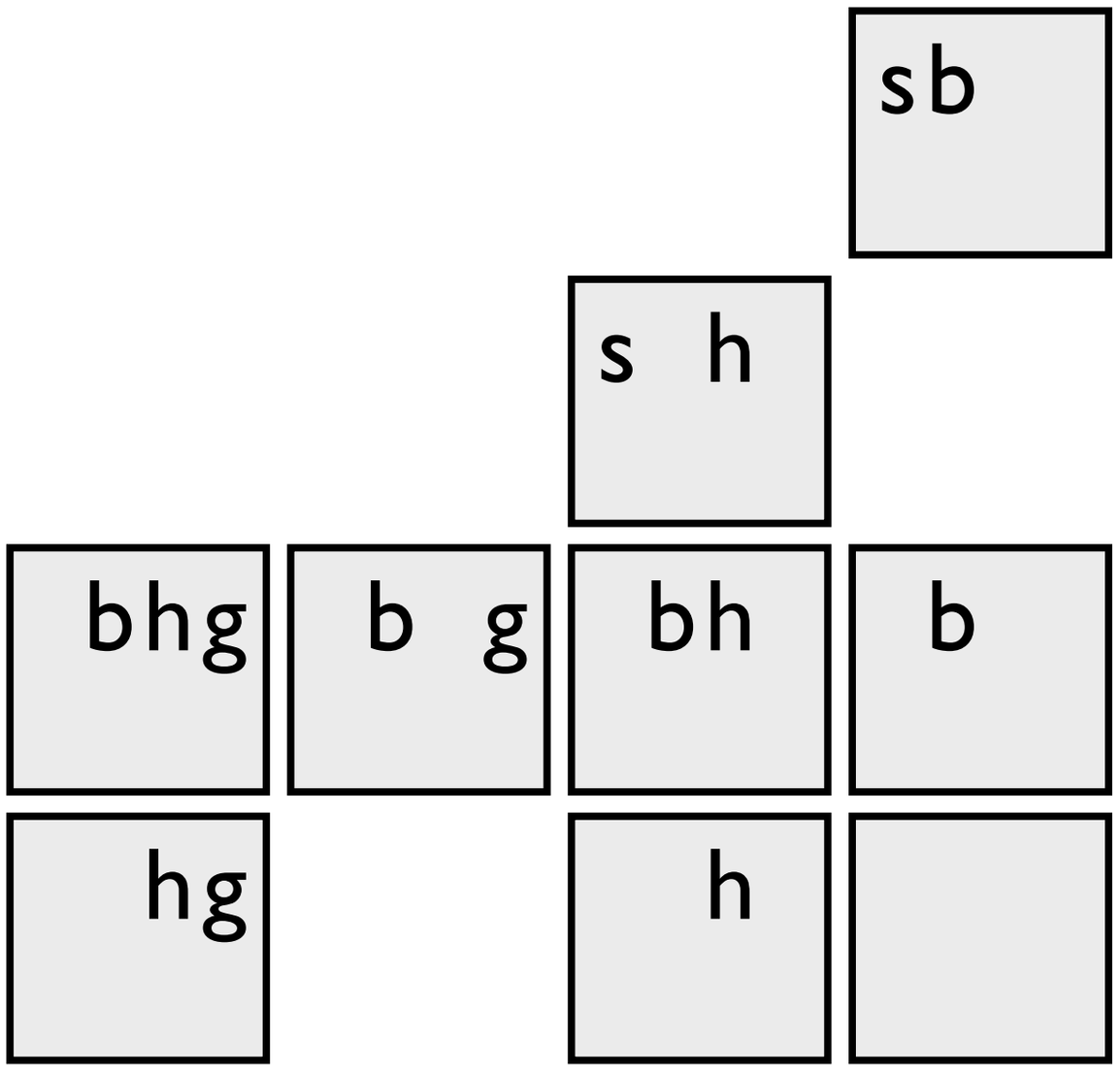
[Fierens et al, PP 12; Shterionov et al]

cProbLog: constraints on possible worlds

```
weight(skis,6).
weight(boots,4).
weight(helmet,3).
weight(gloves,2).

P::pack(Item) :-
  weight(Item,Weight),
  P is 1.0/Weight.

excess(Limit) :- ...
```

```
not excess(10).
pack(helmet) v pack(boots).
```

constraints as first-order logic formulas



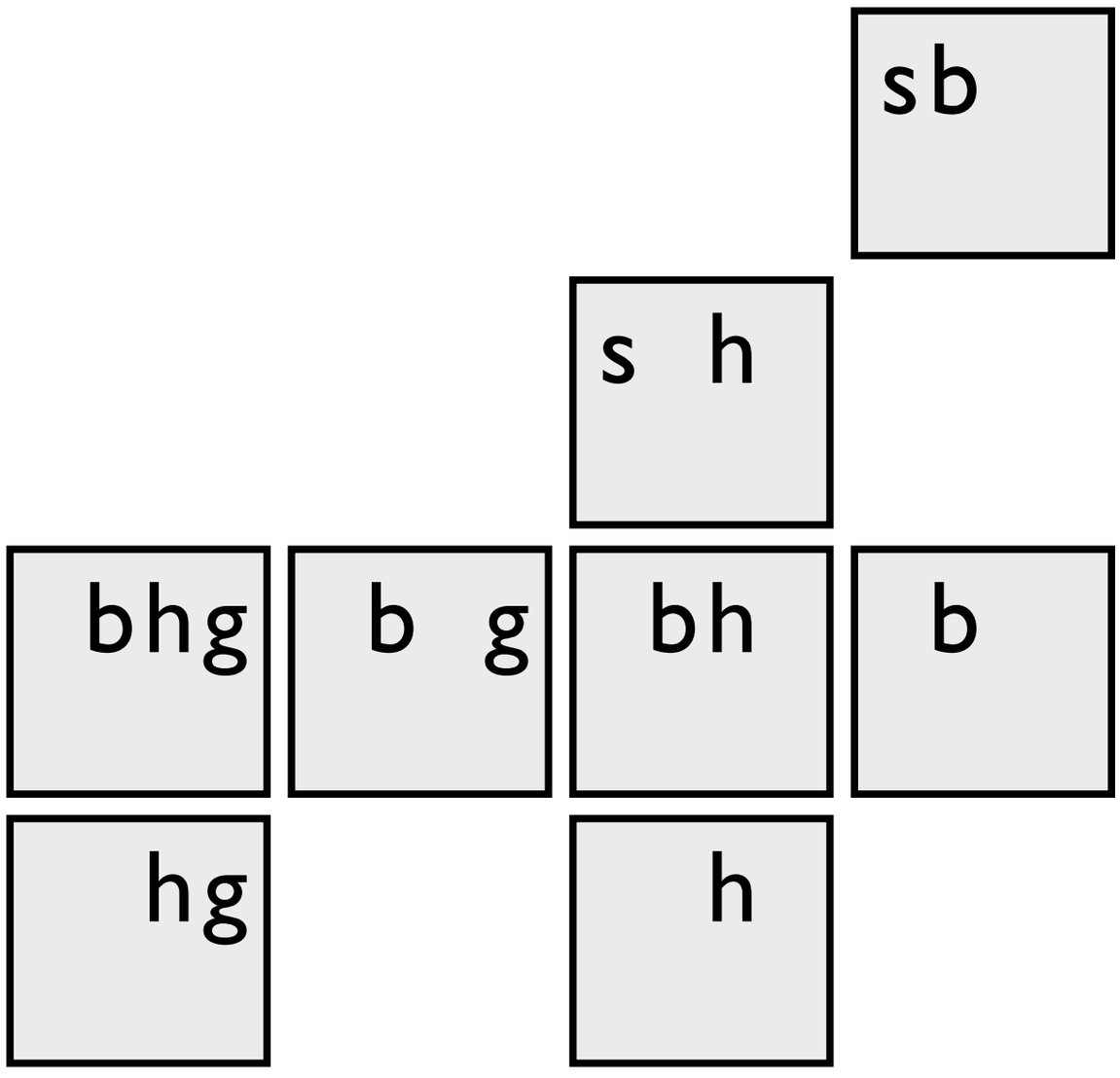
[Fierens et al, PP 12; Shterionov et al]

cProbLog: constraints on possible worlds

```
weight(skis,6).
weight(boots,4).
weight(helmet,3).
weight(gloves,2).

P::pack(Item) :-
  weight(Item,Weight),
  P is 1.0/Weight.

excess(Limit) :- ...
```

```
not excess(10).
pack(helmet) v pack(boots).
```

constraints as first-order logic formulas



[Fierens et al, PP 12; Shterionov et al]

cProbLog: constraints on possible worlds

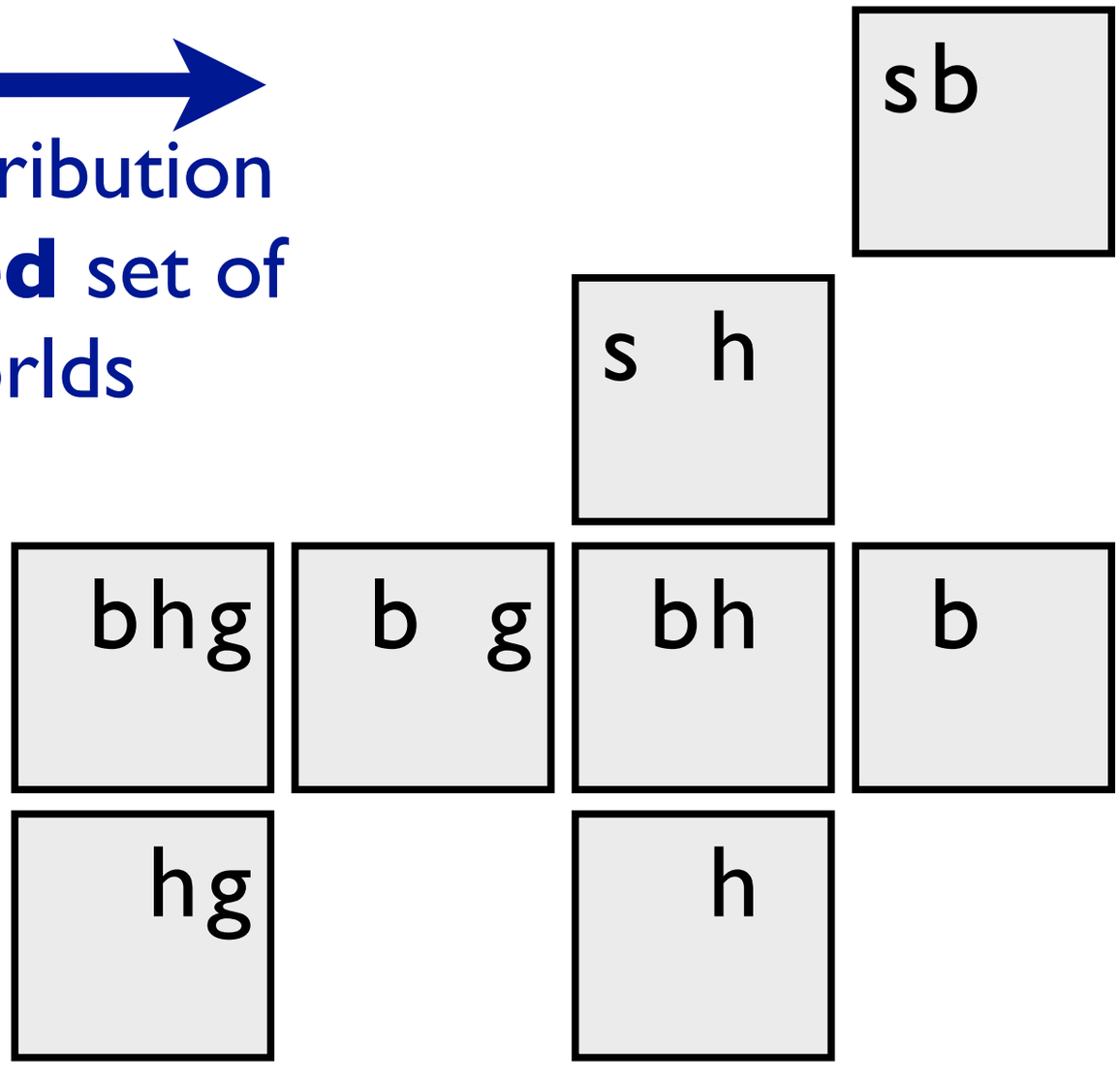
```
weight(skis,6).
weight(boots,4).
weight(helmet,3).
weight(gloves,2).
```

```
P::pack(Item) :-
  weight(Item,Weight),
  P is 1.0/Weight.

excess(Limit) :- ...
```

```
not excess(10).
pack(helmet) v pack(boots).
```

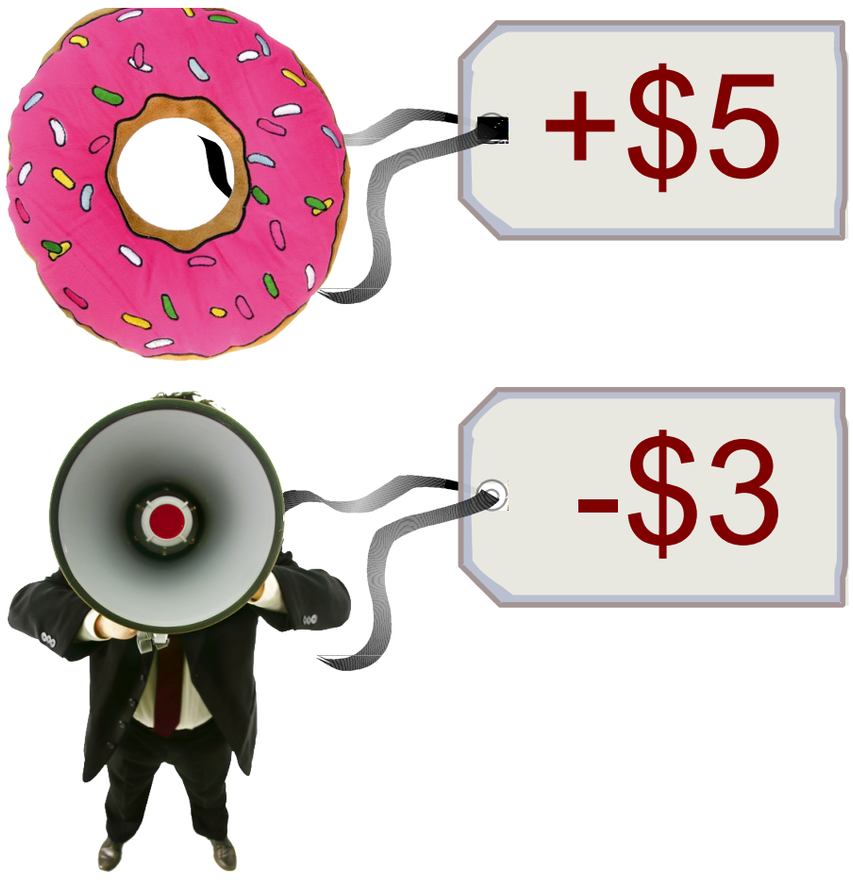
constraints as first-order logic formulas

normalized distribution over **restricted** set of possible worlds

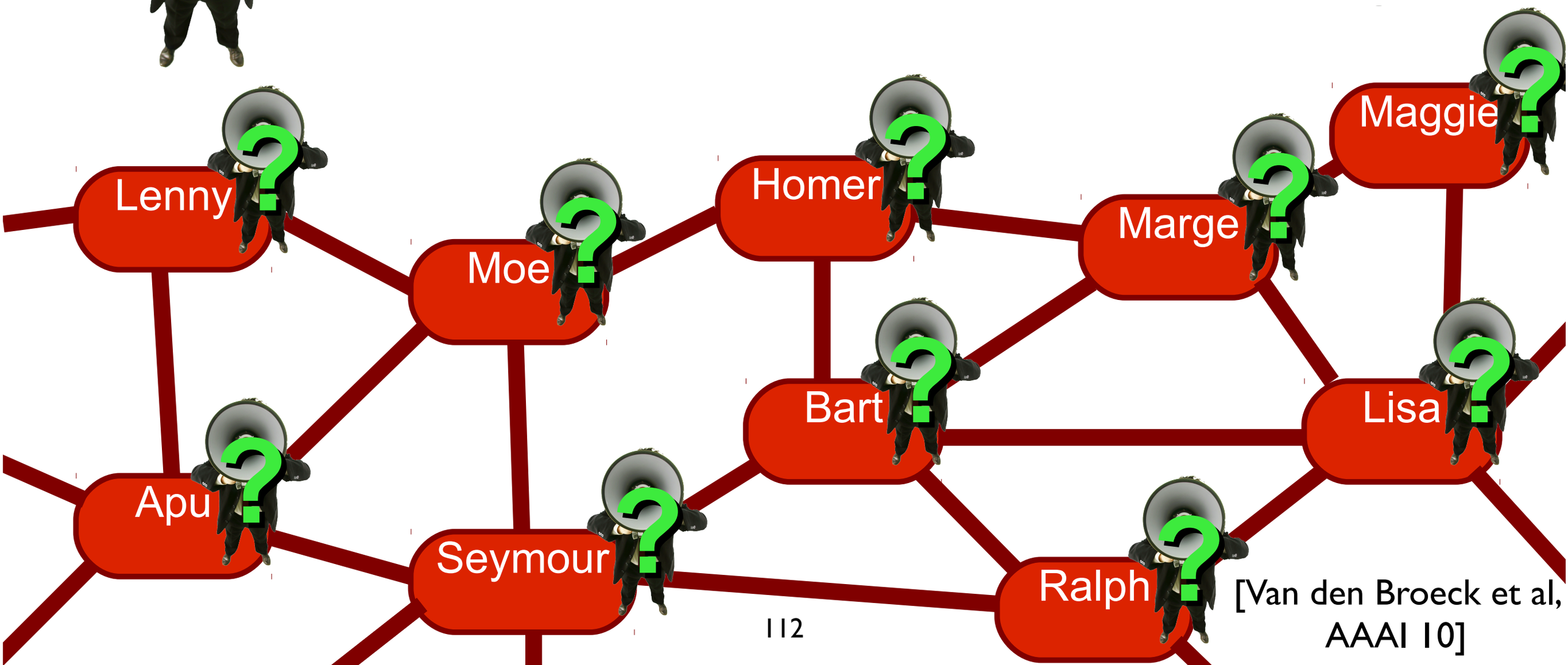


[Fierens et al, PP 12; Shterionov et al]

Viral Marketing

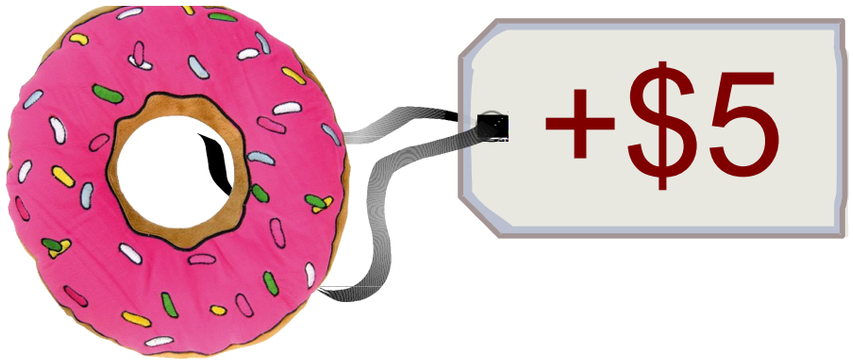


Which advertising strategy maximizes expected profit?

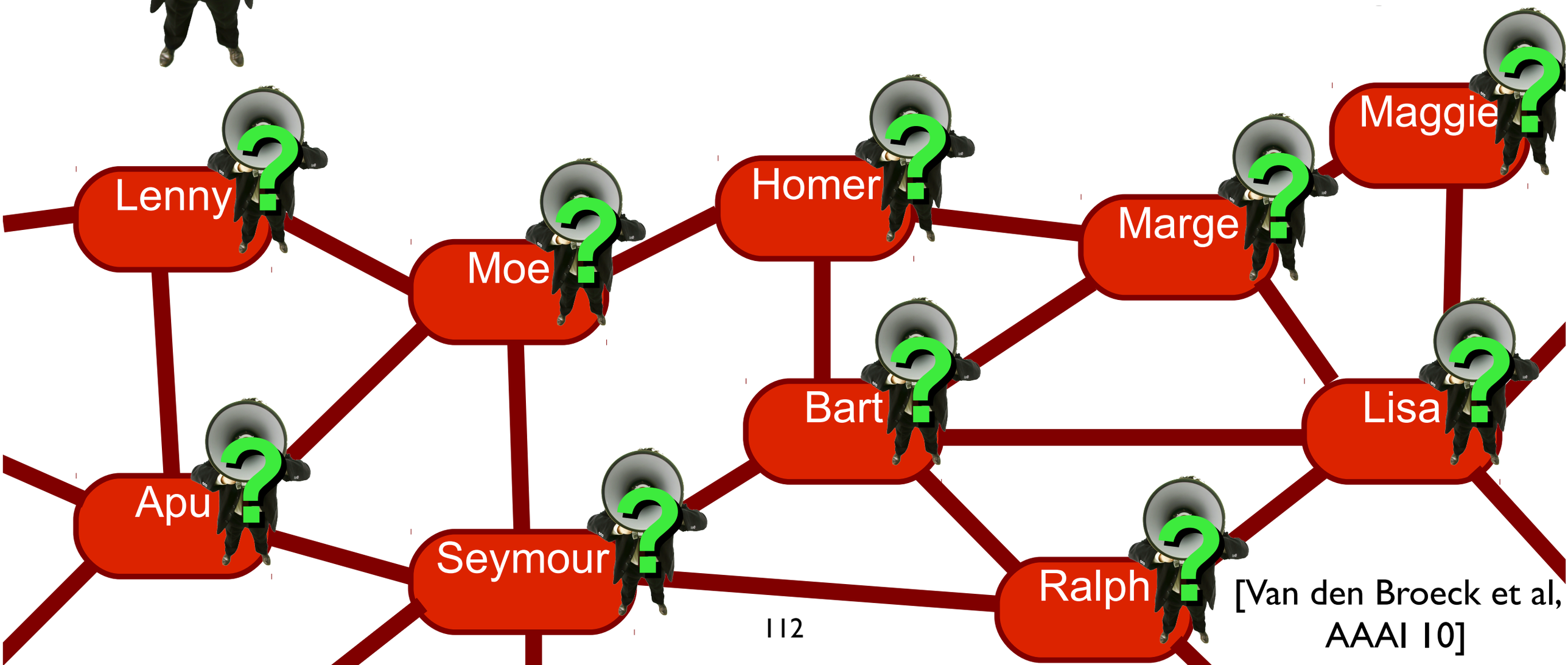


[Van den Broeck et al, AAAI 10]

Viral Marketing

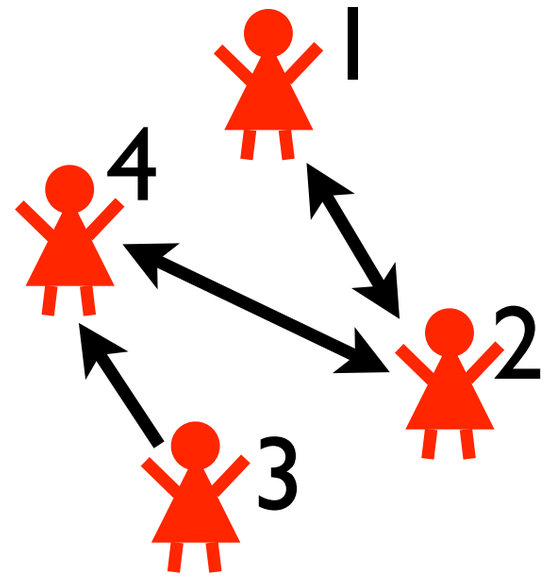


decide truth values of some atoms



[Van den Broeck et al, AAAI 10]

DTProbLog



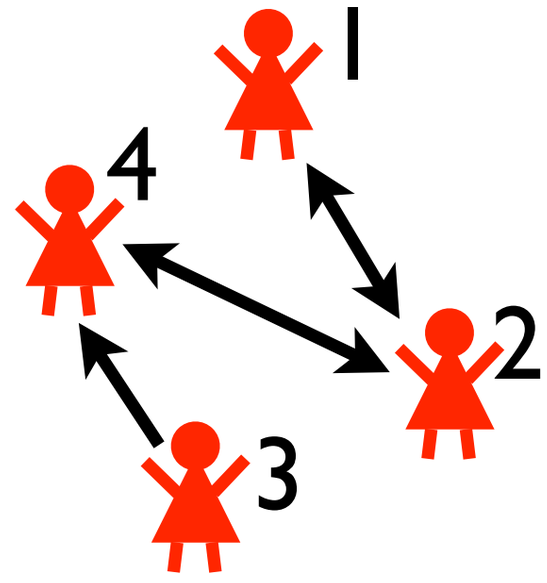

```
person(1).
person(2).
person(3).
person(4).
```

```
friend(1,2).
friend(2,1).
friend(2,4).
friend(3,4).
friend(4,2).
```

DTProbLog

```
? :: marketed(P) :- person(P).
```

decision fact: true or false?




```
person(1).
person(2).
person(3).
person(4).
```

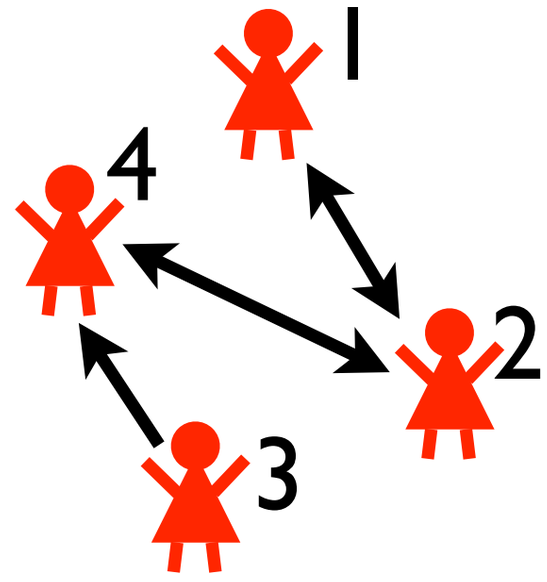
```
friend(1,2).
friend(2,1).
friend(2,4).
friend(3,4).
friend(4,2).
```

DTProbLog

```
? :: marketed(P) :- person(P).
```

```
0.3 :: buy_trust(X,Y) :- friend(X,Y).
0.2 :: buy_marketing(P) :- person(P).

buys(X) :- friend(X,Y), buys(Y), buy_trust(X,Y).
buys(X) :- marketed(X), buy_marketing(X).
```

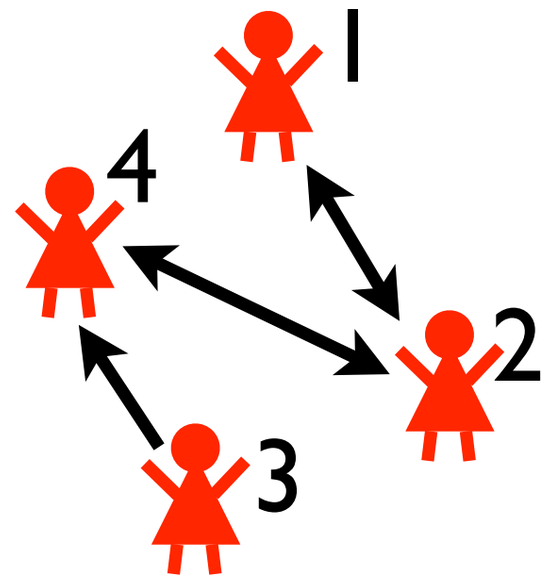
probabilistic facts
+ logical rules




```
person(1).
person(2).
person(3).
person(4).
```

```
friend(1,2).
friend(2,1).
friend(2,4).
friend(3,4).
friend(4,2).
```

DTProbLog

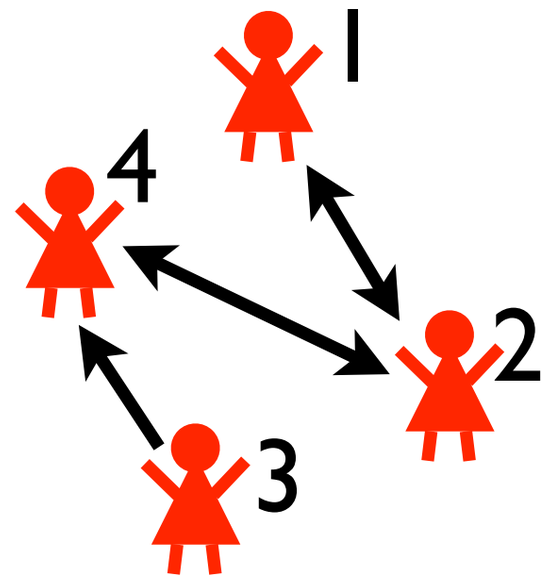



```
? :: marketed(P) :- person(P).

0.3 :: buy_trust(X,Y) :- friend(X,Y).
0.2 :: buy_marketing(P) :- person(P).

buys(X) :- friend(X,Y), buys(Y), buy_trust(X,Y).
buys(X) :- marketed(X), buy_marketing(X).

buys(P) => 5 :- person(P).
marketed(P) => -3 :- person(P).
```

```
person(1).
person(2).
person(3).
person(4).

friend(1,2).
friend(2,1).
friend(2,4).
friend(3,4).
friend(4,2).
```

utility facts: cost/reward if true

DTProbLog

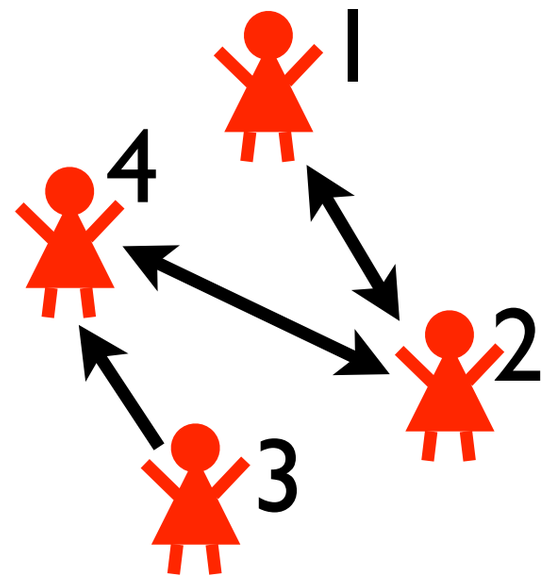



```
? :: marketed(P) :- person(P).

0.3 :: buy_trust(X,Y) :- friend(X,Y).
0.2 :: buy_marketing(P) :- person(P).

buys(X) :- friend(X,Y), buys(Y), buy_trust(X,Y).
buys(X) :- marketed(X), buy_marketing(X).

buys(P) => 5 :- person(P).
marketed(P) => -3 :- person(P).
```

```
person(1).
person(2).
person(3).
person(4).

friend(1,2).
friend(2,1).
friend(2,4).
friend(3,4).
friend(4,2).
```

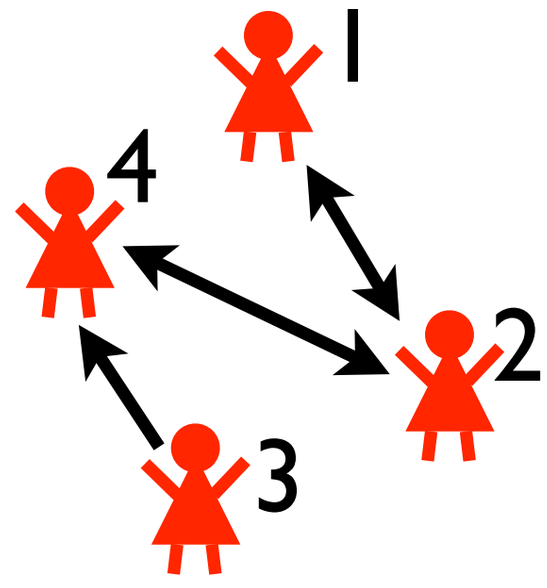
DTProbLog

```
? :: marketed(P) :- person(P).

0.3 :: buy_trust(X,Y) :- friend(X,Y).
0.2 :: buy_marketing(P) :- person(P).

buys(X) :- friend(X,Y), buys(Y), buy_trust(X,Y).
buys(X) :- marketed(X), buy_marketing(X).

buys(P) => 5 :- person(P).
marketed(P) => -3 :- person(P).
```




```
person(1).
person(2).
person(3).
person(4).

friend(1,2).
friend(2,1).
friend(2,4).
friend(3,4).
friend(4,2).
```

DTProbLog

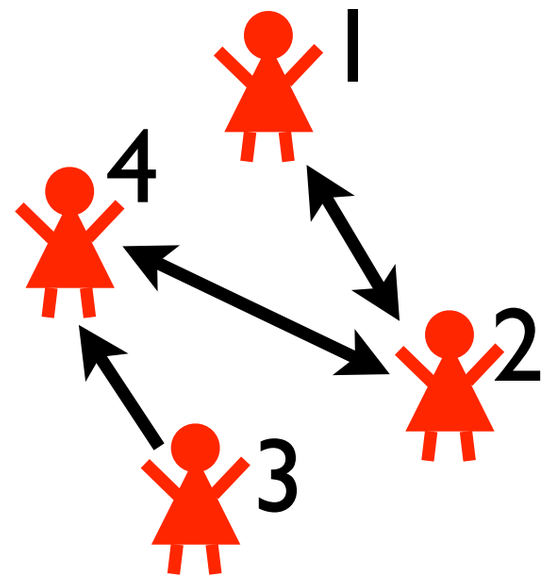



```
? :: marketed(P) :- person(P).

0.3 :: buy_trust(X,Y) :- friend(X,Y).
0.2 :: buy_marketing(P) :- person(P).

buys(X) :- friend(X,Y), buys(Y), buy_trust(X,Y).
buys(X) :- marketed(X), buy_marketing(X).

buys(P) => 5 :- person(P).
marketed(P) => -3 :- person(P).
```

```
person(1).
person(2).
person(3).
person(4).

friend(1,2).
friend(2,1).
friend(2,4).
friend(3,4).
friend(4,2).
```

```
marketed(1)        marketed(3)
```

DTProbLog

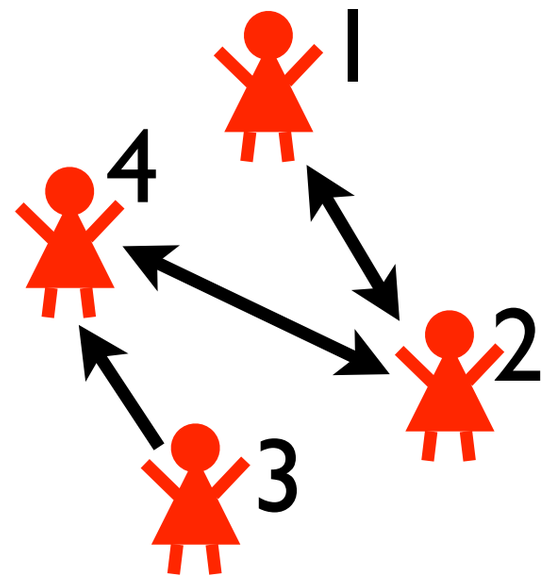



```
? :: marketed(P) :- person(P).

0.3 :: buy_trust(X,Y) :- friend(X,Y).
0.2 :: buy_marketing(P) :- person(P).

buys(X) :- friend(X,Y), buys(Y), buy_trust(X,Y).
buys(X) :- marketed(X), buy_marketing(X).

buys(P) => 5 :- person(P).
marketed(P) => -3 :- person(P).
```

```
person(1).
person(2).
person(3).
person(4).

friend(1,2).
friend(2,1).
friend(2,4).
friend(3,4).
friend(4,2).
```

```
marketed(1)        marketed(3)

    bt(2,1)   bt(2,4)          bm(1)
```

DTProbLog

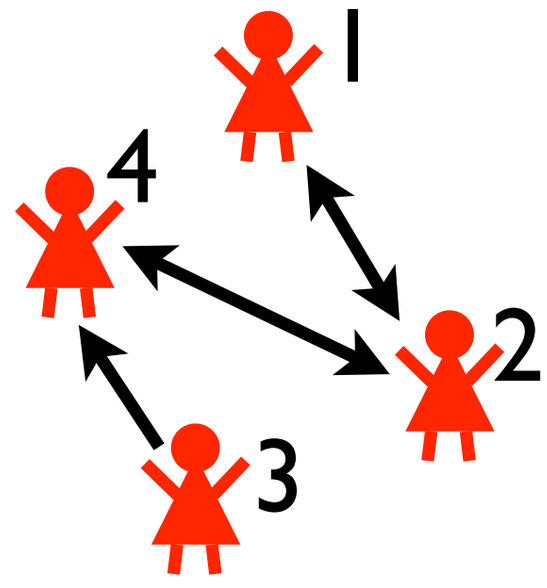



```
? :: marketed(P) :- person(P).

0.3 :: buy_trust(X,Y) :- friend(X,Y).
0.2 :: buy_marketing(P) :- person(P).

buys(X) :- friend(X,Y), buys(Y), buy_trust(X,Y).
buys(X) :- marketed(X), buy_marketing(X).

buys(P) => 5 :- person(P).
marketed(P) => -3 :- person(P).
```

```
person(1).
person(2).
person(3).
person(4).

friend(1,2).
friend(2,1).
friend(2,4).
friend(3,4).
friend(4,2).
```

```
marketed(1)        marketed(3)

    bt(2,1)   bt(2,4)          bm(1)

buys(1)    buys(2)
```

DTProbLog



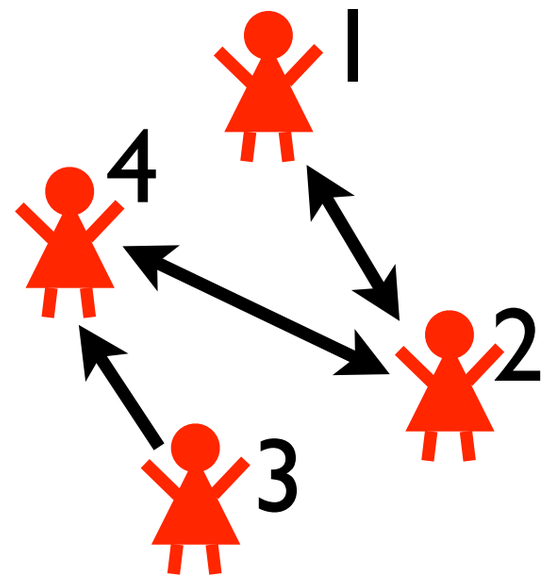

```
? :: marketed(P) :- person(P).

0.3 :: buy_trust(X,Y) :- friend(X,Y).
0.2 :: buy_marketing(P) :- person(P).

buys(X) :- friend(X,Y), buys(Y), buy_trust(X,Y).
buys(X) :- marketed(X), buy_marketing(X).

buys(P) => 5 :- person(P).
marketed(P) => -3 :- person(P).
```

```
person(1).
person(2).
person(3).
person(4).

friend(1,2).
friend(2,1).
friend(2,4).
friend(3,4).
friend(4,2).
```

utility = −3 + −3 + 5 + 5 = 4

probability = 0.0032

DTProbLog




```
? :: marketed(P) :- person(P).

0.3 :: buy_trust(X,Y) :- friend(X,Y).
0.2 :: buy_marketing(P) :- person(P).

buys(X) :- friend(X,Y), buys(Y), buy_trust(X,Y).
buys(X) :- marketed(X), buy_marketing(X).

buys(P) => 5 :- person(P).
marketed(P) => -3 :- person(P).
```

```
person(1).
person(2).
person(3).
person(4).

friend(1,2).
friend(2,1).
friend(2,4).
friend(3,4).
friend(4,2).
```

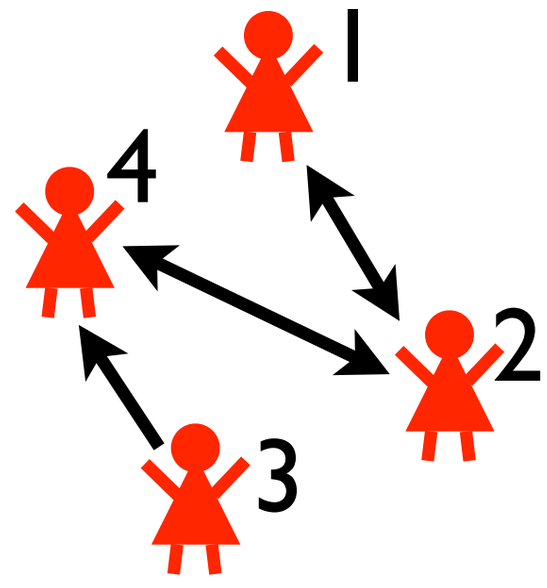
utility = −3 + −3 + 5 + 5 = 4

probability = 0.0032

```
marketed(1)        marketed(3)

   bt(2,1)   bt(2,4)          bm(1)

buys(1)    buys(2)
```

world contributes 0.0032×4 to expected utility of strategy

DTProbLog

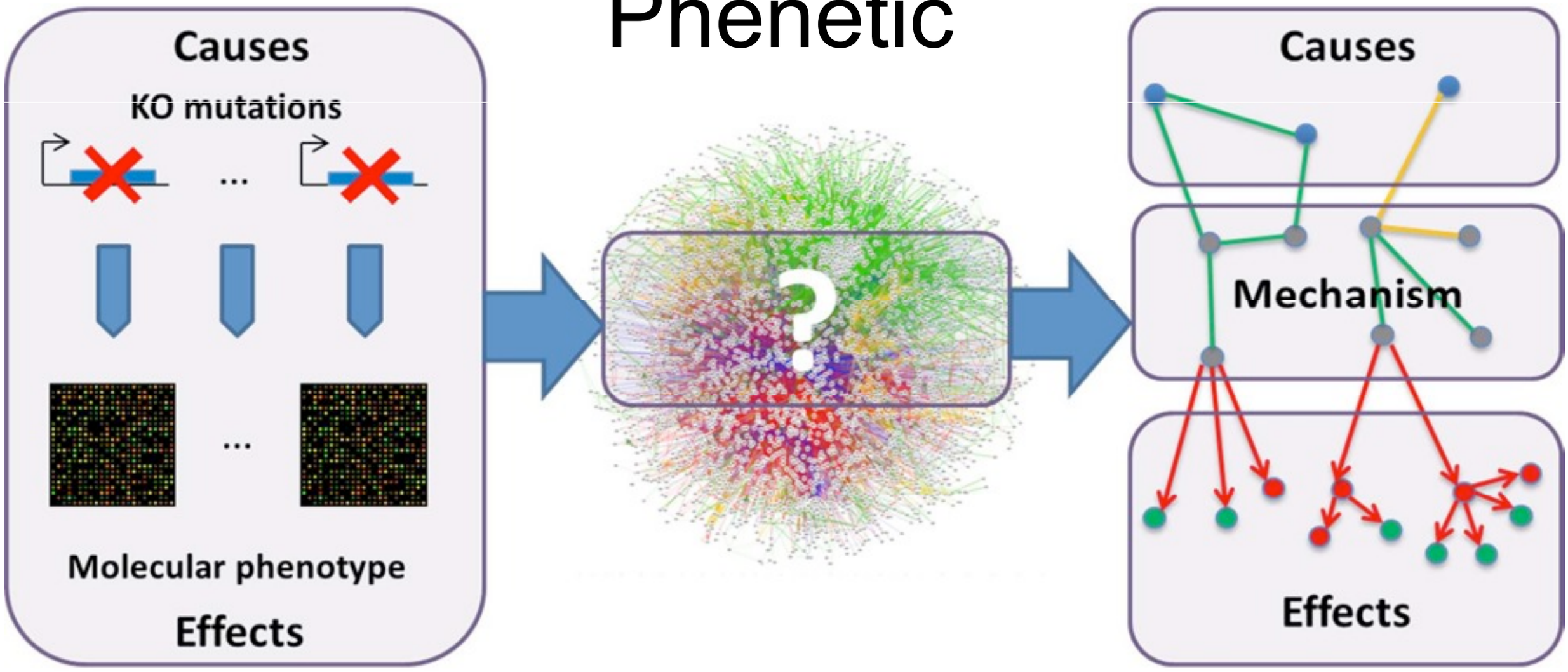



```
? :: marketed(P) :- person(P).

0.3 :: buy_trust(X,Y) :- friend(X,Y).
0.2 :: buy_marketing(P) :- person(P).

buys(X) :- friend(X,Y), buys(Y), buy_trust(X,Y).
buys(X) :- marketed(X), buy_marketing(X).

buys(P) => 5 :- person(P).
marketed(P) => -3 :- person(P).
```

```
person(1).
person(2).
person(3).
person(4).

friend(1,2).
friend(2,1).
friend(2,4).
friend(3,4).
friend(4,2).
```

task: find strategy that maximizes expected utility
solution: using ProbLog technology

Phenetic



- Causes: Mutations
  - All related to similar phenotype
- Effects: Differentially expressed genes
- 27 000 cause effect pairs

- Interaction network:
  - 3063 nodes
    - Genes
    - Proteins
  - 16794 edges
    - Molecular interactions
    - Uncertain

- Goal: connect causes to effects through common subnetwork
  - = Find mechanism
- Techniques:
  - DTProbLog
  - Approximate inference

[De Maeyer et al., Molecular Biosystems 13]

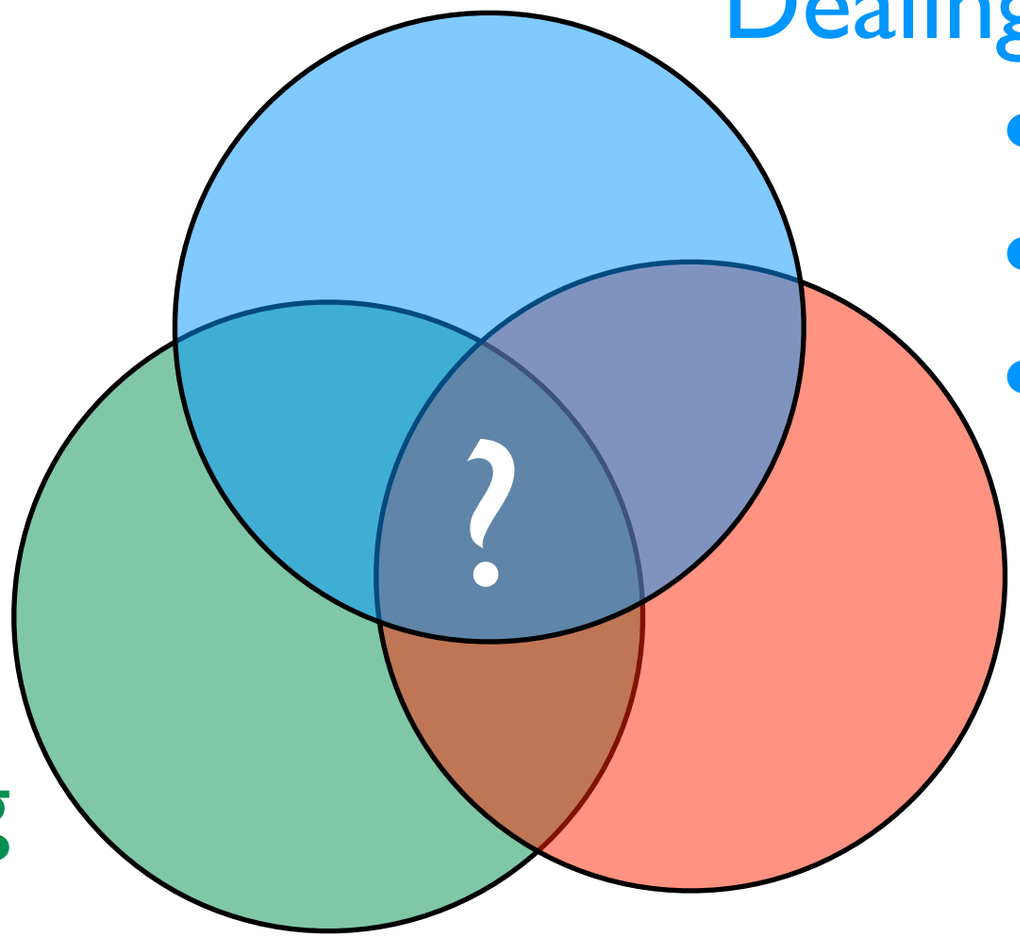
Roadmap

- Modeling (with detours to related work)
- Reasoning (and a bit of learning)
- Language extensions

A key question in AI:

Reasoning with relational data

- logic
- databases
- programming
- ...



Dealing with uncertainty

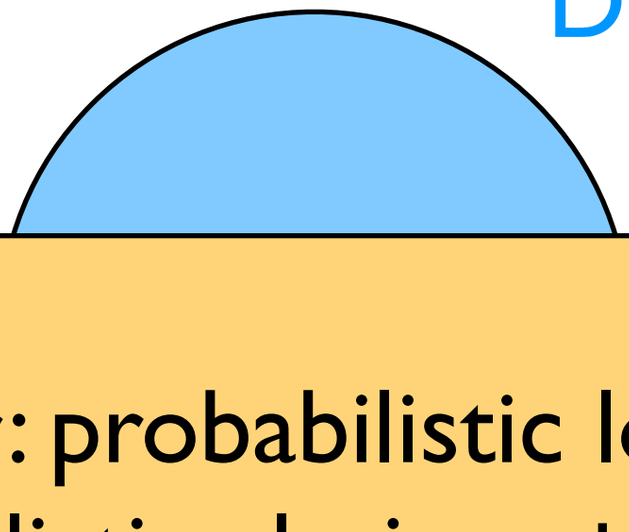
- probability theory
- graphical models
- ...

Learning

- parameters
- structure

Statistical relational learning, probabilistic logic learning, probabilistic programming, ...

A key question in AI:



Dealing with uncertainty
- probability theory

Reas
relat
- log
- da
- pr
- ...

dels

- Our answer: probabilistic logic programming
  = probabilistic choices + logic program
- Many languages, systems, applications, ...
- ... and much more to do!

structure

Statistical relational learning, probabilistic logic learning, probabilistic programming, ...

PLP Systems

- **PRISM** http://sato-www.cs.titech.ac.jp/prism/
- **ProbLog2** http://dtai.cs.kuleuven.be/problog/
- Yap Prolog http://www.dcc.fc.up.pt/~vsc/Yap/ includes
  - **ProbLog1**
  - **cplint** https://sites.google.com/a/unife.it/ml/cplint
  - **CLP(BN)**
  - **LP2**
- **PITA** in XSB Prolog http://xsb.sourceforge.net/
- **AILog2** http://artint.info/code/ailog/ailog2.html
- **SLPs** http://stoics.org.uk/~nicos/sware/pepl
- **contdist** http://www.cs.sunysb.edu/~cram/contdist/
- **DC** https://code.google.com/p/distributional-clauses
- **WFOMC** http://dtai.cs.kuleuven.be/ml/systems/wfomc

References

Bach SH, Broecheler M, Getoor L, O'Leary DP (2012) Scaling MPE inference for constrained continuous Markov random fields with consensus optimization. In: Proceedings of the 26th Annual Conference on Neural Information Processing Systems (NIPS-12)

Bryant RE (1986) Graph-based algorithms for Boolean function manipulation. IEEE Transactions on Computers 35(8):677–691

Cussens J (2001) Parameter estimation in stochastic logic programs. Machine Learning 44(3):245–271

De Maeyer D, Renkens J, Cloots L, De Raedt L, Marchal K (2013) Phenetic: network-based interpretation of unstructured gene lists in e. coli. Molecular BioSystems 9(7):1594–1603

De Raedt L, Kimmig A (2013) Probabilistic programming concepts. CoRR abs/1312.4328

De Raedt L, Kimmig A, Toivonen H (2007) ProbLog: A probabilistic Prolog and its application in link discovery. In: Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI-07)

Fierens D, Van den Broeck G, Bruynooghe M, De Raedt L (2012) Constraints for probabilistic logic programming. In: Proceedings of the NIPS Probabilistic Programming Workshop

Fierens D, Van den Broeck G, Renkens J, Shterionov D, Gutmann B, Thon I, Janssens G, De Raedt L (2014) Inference and learning in probabilistic logic programs using weighted Boolean formulas. Theory and Practice of Logic Programming (TPLP) FirstView

Goodman N, Mansinghka VK, Roy DM, Bonawitz K, Tenenbaum JB (2008) Church: a language for generative models. In: Proceedings of the 24th Conference on Uncertainty in Artificial Intelligence (UAI-08)

Gutmann B, Thon I, De Raedt L (2011a) Learning the parameters of probabilistic logic programs from interpretations. In: Proceedings of the 22nd European Conference on Machine Learning (ECML-11)

Gutmann B, Thon I, Kimmig A, Bruynooghe M, De Raedt L (2011b) The magic of logical inference in probabilistic programming. Theory and Practice of Logic Programming (TPLP) 11((4–5)):663–680

Huang B, Kimmig A, Getoor L, Golbeck J (2013) A flexible framework for probabilistic models of social trust. In: Proceedings of the International Conference on Social Computing, Behavioral-Cultural Modeling, and Prediction (SBP-13)

Kimmig A, Demoen B, De Raedt L, Santos Costa V, Rocha R (2011) On the implementation of the probabilistic logic programming language ProbLog. Theory and Practice of Logic Programming (TPLP) 11:235–262

Milch B, Marthi B, Russell SJ, Sontag D, Ong DL, Kolobov A (2005) Blog: Probabilistic models with unknown objects. In: Proceedings of the 19th International Joint Conference on Artificial Intelligence (IJCAI-05)

Moldovan B, De Raedt L (2014) Occluded object search by relational affordances. In: IEEE International Conference on Robotics and Automation (ICRA-14)

Moldovan B, Moreno P, van Otterlo M, Santos-Victor J, De Raedt L (2012) Learning relational affordance models for robots in multi-object manipulation tasks. In: IEEE International Conference on Robotics and Automation (ICRA-12)

Muggleton S (1996) Stochastic logic programs. In: De Raedt L (ed) Advances in Inductive Logic Programming, IOS Press, pp 254–264

Nitti D, De Laet T, De Raedt L (2013) A particle filter for hybrid relational domains. In: Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS-13)

Nitti D, De Laet T, De Raedt L (2014) Relational object tracking and learning. In: IEEE International Conference on Robotics and Automation (ICRA), June 2014

Pfeffer A (2001) IBAL: A probabilistic rational programming language. In: Proceedings of the 17th International Joint Conference on Artificial Intelligence (IJCAI-01)

Pfeffer A (2009) Figaro: An object-oriented probabilistic programming language. Tech. rep., Charles River Analytics

Sato T (1995) A statistical learning method for logic programs with distribution semantics. In: Proceedings of the 12th International Conference on Logic Programming (ICLP-95)

Sato T, Kameya Y (2001) Parameter learning of logic programs for symbolic-statistical modeling. J Artif Intell Res (JAIR) 15:391–454

Sato T, Kameya Y (2008) New advances in logic-based probabilistic modeling by prism. In: Probabilistic Inductive Logic Programming, pp 118–155

Skarlatidis A, Artikis A, Filiopou J, Paliouras G (2014) A probabilistic logic programming event calculus. Theory and Practice of Logic Programming (TPLP) FirstView

Suciu D, Olteanu D, Ré C, Koch C (2011) Probabilistic Databases. Synthesis Lectures on Data Management, Morgan & Claypool Publishers

Thon I, Landwehr N, De Raedt L (2011) Stochastic relational processes: Efficient inference and applications. Machine Learning 82(2):239–272

Van den Broeck G, Thon I, van Otterlo M, De Raedt L (2010) DTProbLog: A decision-theoretic probabilistic Prolog. In: Proceedings of the 24th AAAI Conference on Artificial Intelligence (AAAI-10)

Vennekens J, Verbaeten S, Bruynooghe M (2004) Logic programs with annotated disjunctions. In: Proceedings of the 20th International Conference on Logic Programming (ICLP-04)

Wang WY, Mazaitis K, Cohen WW (2013) Programming with personalized pagerank: a locally groundable first-order probabilistic logic. In: Proceedings of the 22nd ACM International Conference on Information and Knowledge Management (CIKM-13)